68' MICRO JOURNAL

Australia A $4.75 New Zealand NZ $6.50
Singapore S $9.45 Hong Kong H $23.50
Malaysia M $9.45 Sweden 30,-SEK

$2.95 USA

Motorola VME-MACINTOSH-S 50 & Other 68XXX Systems
6809 68008 68000 68010 68020 68030
The Magazine for Motorola CPU Devices
OS-9 — For Over a Decade! A User Contributor Journal — FLEX SK*DOS

This Issue:

And Lots More!

VOLUME IX ISSUE VI ● Devoted to the 68XXX User ● June 1987

"Small Computers Doing Big Things"

GMX INC. 1337 WEST 37th PLACE • CHICAGO, ILLINOIS 60609 • (312) 927-5510 • TWX 910-221-4055

GMX MICRO 20 PRICE LIST

MICRO 20 (12.5 MHz) W/1 SAB...............$2565.00
MICRO 20 (16.67 MHz) W/1 SAB..............$2895.00
MICRO 20 (20 MHz) W/1 SAB.................$3295.00

OPTIONAL PARTS AND ACCESSORIES

68881RC12.................................$ 295.00
68881RC16.................................$ 395.00
MOTOROLA 68020 USERS MANUAL...............$ 18.00
MOTOROLA 68881 USERS MANUAL...............$ 18.00

SBC ACCESSORY PACKAGE (M20-AP)......$1690.00
The package includes a PC-style cabinet with a custom backpanel, a Xebec 1410A hard disk controller, a 25 Megabyte (unformatted) hard disk, a 5 1/4" DSDD 80 track floppy disk drive, a 150 watt power supply, cooling fan, panel mounted reset and abort switches and all necessary internal cabling
BACK PANEL PLATE (BPP-PC) For Above.......$ 44.00
2nd 5"80 FLOPPY & CABLES FOR M20-AP, ADD..$ 250.00
SECOND 25MB HARD DISK & CABLES, ADD.......$ 780.00
TO SUBSTITUTE 85MB HD FOR 25MB HD, ADD....$1800.00

I/O EXPANSION BOARDS

16 PORT SERIAL CARD ONLY (SBC-16S)........$335.00
The SBC-16S extends the I/O capabilities of the GMX Micro-20 68020 Single-board Computer by adding sixteen asynchronous serial I/O ports. By using two SBC-16S boards, a total of thirty-six serial ports are possible.

RS232 Adapter(SAB-25,SAB-9D or RJ-45).....$165.00
The board provides level-shifting between TTL level and standard RS-232 signal levels for up to 4 serial I/O ports.

60 LINE PARALLEL I/O CARD (SBC-60P).......$398.00
The GMX SBC-60P uses three 68230 Parallel Interface/Timers (PI/Ts) to provide up to forty-eight parallel I/O lines. The I/O lines are buffered in six groups of eight lines each, with separate buffer direction control for each group. Buffer direction can be fixed by hardware jumpers, or can be software programmable for bidirectional applications.

PROTOTYPING BOARD (SBC-WW)................$75.00
The SBC-WW provides a means of developing and testing custom I/O interface designs for the GMX Micro-20 68020 Single-board Computer. The board provides areas for both DIP (Dual Inline Package) and PGA (Pin Grid Array) devices, and a pre-wired memory area for up to 512K bytes of dynamic RAM.

I/O BUS ADAPTER (SBC-BA)..................$195.00
The SBC-BA provides an interface between the GMX Micro-20 68020 Single-board Computer and the Motorola Input/Output Channel (I/O bus). With the I/O bus, up to sixteen off-the-shelf or custom peripheral devices (I/O modules) can be connected to the GMX Micro-20.

ARCNET LAN board w/o Software (SBC-AN)....$475.00
The SBC-AN provides an interface between the GMX Micro-20 68020 Single-board Computer and the ARCNET modified token-passing Local Area Network (LAN) originally developed by Datapoint Corp. The ARCNET is a baseband network with a data transmission rate of 2.5 Megabits/second. The standard transmission media is a single 93 ohm RG-62/U coaxial cable. Fiber optic versions are available as an option.

OS9 LAN Software Drivers for SBC-AN.......$ 120.00

GMX MICRO 20 SOFTWARE

020 BUG UPDATE - PROMS & MANUAL...........$ 150.00

THESE 68020 OPERATING SYSTEMS ARE PRICED WHEN PURCHASED WITH THE MICRO-20. PLEASE ADD $150.00 IF PURCHASED LATER FOR THE UPDATED PROMS AND MANUALS.

OS9

OS9/68020 PROFESSIONAL PAK................$ 850.00
Includes O.S., "C", EDITOR, ASSEMBLER, DEBUGGER, development utilities, 68881 support.
Other Software for OS-9/68020
BASIC (included in PERSONAL PAK)..........$ 200.00
C COMPILER (included in PROFESSIONAL PAK).$ 500.00
PASCAL COMPILER...........................$ 500.00

UNIFLEX

UniFLEX (when ordered with Board).........$ 450.00
UniFLEX WITH REAL-TIME ENHANCEMENTS.......$1000.00
Other Software for UniFLEX
UniFLEX BASIC W/PRECOMPILER...............$ 300.00
UniFLEX C COMPILER........................$ 350.00
UniFLEX COBOL COMPILER....................$ 750.00
UniFLEX SCREEN EDITOR.....................$ 150.00
UniFLEX TEXT PROCESSOR....................$ 200.00
UniFLEX SORT/MERGE PACKAGE................$ 200.00
UniFLEX VSAM MODULE.......................$ 100.00
UniFLEX UTILITIES PACKAGE I...............$ 200.00
UniFLEX PARTIAL SOURCE LICENSE............$1000.00

GMX EXCLUSIVE VERSIONS,CUSTOMIZED FOR THE MICRO-20, OF THE BELOW LANGUAGES AND SOFTWARE ARE ALSO AVAILABLE FROM GMX.

ABSOFT FORTRAN (UniFLEX)..................$1500.00

SCULPTOR (specify UniFLEX or OS9).........$ 995.00

FORTH (OS9)...............................$ 595.00

DYNACALC (specify UniFLEX or OS9).........$ 300.00

GMX DOES NOT GUARANTEE PERFORMANCE OF ANY GMX SYSTEMS, BOARDS OR SOFTWARE WHEN USED WITH OTHER MANUFACTURES PRODUCT.

ALL PRICES ARE F.O.B. CHICAGO

GMX, Inc. reserves the right to change pricing, terms, and products specifications at any time without further notice.

TO ORDER BY MAIL: SEND CHECK OR MONEY ORDER OR USE YOUR VISA OR MASTER CHARGE. Please allow 3 weeks for personal checks to clear. U.S. orders add $5 handling if under $200.00. Foreign orders add $10 handling if order is under $200.00. Foreign orders over $200.00 will be shipped via Emery Air Freight COLLECT, and we will charge no handling. All orders must be prepaid in U.S. funds. Please note that foreign checks have been taking about 8 weeks for collection so we would advise wiring money, or checks drawn on a bank account in the U.S. Our bank is the Continental Illinois National Bank of Chicago, 231 S. LaSalle Street, Chicago, IL 60693, account number 73-32033.

CONTACT GMX FOR MORE INFORMATION ON THE ABOVE PRODUCTS

GMX STILL SELLS GIMIX S50 BUS SYSTEMS, BOARDS & PARTS. CONTACT GMX FOR COMPLETE PRICE LIST.

A Member of the CPI Family

68 Micro Journal

10 Years of Dedication to Motorola CPU Users

6800 6809 68000 68010 68020

Contents

68 MICRO JOURNAL

"Contribute Nothing - Expect Nothing" DMW

COMPUTER PUBLISHING, INC.

"Over a Decade of Service"

"World Wide"

68 MICRO JOURNAL
Computer Publishing Center
5900 Cassandra Smith Road
PO Box 849
Hixson, TN 37343

Phone (615) 842-4600 Telex 510 600-6630

68 Micro Journal is published 12 times a year by Computer Publishing Inc. Second Class Postage paid ISSN 0194-5025 at Hixson, TN. and additional entries. Postmaster: send form 3597 to 68 Micro Journal, POB 849, Hixson, TN 37343.

Subscription Rates

1 Year $24.50 USA, Canada & Mexico $34.00 a year. Others add $12.00 a year surface, $48.00 a year Airmail, USA funds. 2 years $42.50, 3 years $64.50 plus additional postage for each additional year.

Items or Articles for Publication

Articles submitted for publication must include authors name, address, telephone number, date and a statement that the material is original and the property of the author. Articles submitted should be on diskette, OS-9, SK*DOS, FLEX, Macintosh or MS-DOS. All printed items should be dark type and satisfactory for photo-reproduction. No blue ink! No hand written articles - please! Diagrams o.k.

Please - do not format with spaces any text indents, charts, etc. (source listing o.k.). We will edit in all formatting. Text should fall flush left and use a carriage return only to indicate a paragraph end.

Letters & Advertising Copy

Letters to the Editor should be the original copy, signed! Letters of grip as well as praise are acceptable. *We reserve the right to reject any letter or advertising material, for any reason we deem advisable.* Advertising Rates: Commercial please contact 68 Micro Journal Advertising Department. Classified advertising must be non-commercial. Minimum of $15.50 for first 15 words. Add $.60 per word thereafter. No classifieds accepted by telephone.

MUSTANG-020 Super SBC™

DATA-COMP proudly presents the first Under $5000 "SUPER MICRO".

The MUSTANG-020™

CPI
Mustang-020™
A DATA-COMP
Hi-Speed
Product

MUSTANG-020™

The MUSTANG-020 68020 SBC provides a powerful, compact, 32 bit computer system featuring the "state of the art" Motorola 68020 "super" micro-processor. It comes standard with 2 megabyte of high-speed SIP dynamic RAM, serial and parallel ports, floppy disk controller, a SASI hard disk interface for intelligent hard disk controllers and a battery backed-up time-of-day clock. Provisions are made for the super powerful Motorola MC68881 floating point math co-processor, for heavy math and number crunching applications. An optional network interface uses one serial (four (4) standard, expandable to 20) as a 125/bit per second network channel. Supports as many as 32 nodes.

The MUSTANG-020 is ideally suited to a wide variety of applications. It provides a cost effective alternative to the other MC68020 systems now available. It is an excellent introductory tool to the world of hi-power, hi-speed new generation "super nucros". In practical applications it has numerous applications, ranging from scientific to education. It is already being used by government agencies, labs, universities, business and practically every other critical applications center, worldwide, where true multi-user, multi-tasking needs exist. The MUSTANG-020 is UNIX C level V compatible. Whe e low cost and power is a must, the MUSTANG-020 is the answer, as many have discovered. Proving that price is not the standard for quality!

As a software development station, a general purpose scientific or small to medium business computer, or a super efficient real-time controller in process control, the MUSTANG-020 is the cost effective choice. With the optional MC68881 floating point math co-processor installed, it has the capability of systems costing many times over it's total acquisition cost.

With the DATA-COMP "total package", consisting of a heavy duty metal cabinet, switching power supply with rf/line by-passing, 5 inch DS/DD 80 track floppy, Xebec hard disk controller, 25 megabyte winchester hard disk, four serial RS-232 ports and a UNIX C level V compatible multi-tasking, multi-user operating system, the price is under $5000, w/12.5 megahertz system clock (limited time offer). Most all popular high level languages are available at very reasonable cost. The system is expandable to 20 serial ports, at a cost of less than $65 per port, in multiples of 8 port expansion options.

The system SBC fully populated, quality tested, with 4 serial ports pre-wired and board mounted is available for less that $3000. Quantity discounts are available for OEM and special applications, in quantity. All that is required to bring to complete "system" standards is a cabinet, power supply, disks and operating system. All these are available as separate items from DATA-COMP.

A special version of the Motorola 020-BUG is installed on each board. 020-BUG is a ROM based bebugger package with facilities for downloading and executing user programs from a host system. It includes commands for display and modification of memory, breakpoint capabilities, a powerful assembler/disassemble and numerous system diagnostics. Various 020-BUG system routines, such as I/O handlers are available for user programs.

Normal system speed is 3-4.5 MIPS, with burst up to 10 MIPS, at 16.6 megahertz. Intelligent I/O available for some operating systems.

Hands-on "actual experience sessions", before you buy, are available from DATA-COMP. Call or write for additional information or pricing.

DATA-COMP
Installed Systems World-Wide
OVER 16 YEARS OF DEDICATED QUALITY

CPI
A Division of
Computer Publishing, Inc.
5900 Cassandra Smith Road
Hixson, Tn 37343
Telephone 615 842-4600
Telex 810 600-6630

PAT - JUST

PAT

With 'C' Source

$229.00

All OS-9 68XXX Systems

service For Over a Decade! 1,200 Titles +

A Division of CPI

S.E. Media

5900 Cassandra Smith Rd.
Hixson, Tn 37343
Telephone 615 842-6809
Telex 510 600-6630

PAT FROM S. E. MEDIA -- A FULL FEATURED SCREEN ORIENTED TEXT EDITOR with all the best of PIE. For those who swore by and loved **PIE**, this is for **YOU! All PIE features & much more!** Too many features to list. And if you don't like ours, change or add your own. **C source included.** Easily configured to your CRT terminal, with special configuration section. No sweat!

68008 - 68000 - 68010 - 68020 OS-9 68K $229.00

COMBO PAT/JUST

Special $249.00

JUST

JUST from S. E. MEDIA - - Text formatter written by Ron Anderson; for dot matrix printers, provides many unique features. Output formatted to the display. User configurable for adapting to other printers. Comes set-up for Epson MX80 with Graflex. Up to 10 imbedded printer control commands. Compensates for double width printing. Includes normal line width, page numbering, margin, indent, paragraph, space, vertical skip lines, page length, centering, fill, justification, etc. Use with **PAT** or any other text editor. The ONLY stand alone text processor for the 68XXX OS-9 68K, that we have seen. And at a very **LOW PRICE!** Order from: S.E. MEDIA - see catalog this issue.

68008 - 68000 - 68010 - 68020 OS-9 68K
With 'C' source $79.95

The C Programmers Reference Source. Always Right On Target!

C User Notes

A Tutorial Series

By: Dr. E. M. 'Bud' Pass
1454 Latta Lane N.W.
Conyers, GA 30207
404 483-1717/4570
Computer Systems Consultants

This chapter discusses the C Users' Group program library, which has a large number of C programs and functions of potential interest to many C programmers. It also discusses the problems associated with floating-point roundoff, providing considerations for controlling errors caused by the use of floating-point arithmetic.

C USERS' GROUP PROGRAM LIBRARY

The C Users' Group supports a public-domain library of public-domain software written in the C language. There are over 100 volumes of programs in this library, currently available for eight dollars each to members of the C Users' Group. A catalog describing every volume in detail is available for ten dollars to members. A newsletter describing recent additions is also available.

Many of the volumes are specific to particular environments and would require a significant amount of effort to modify them for use in another environment, or would be useless on other computers or operating systems. This reduces the potential usefulness of many of the programs and functions on the volumes.

Another problem with the use of this library concerns the disk formats available. They are readily available in either 8" SSSD CP/M format for CP/M software or 5.25" MSDOS format for most other software. They are available in only other formats which are writable by members of the users' group. None of the standard OS/9, FLEX, UNIFLEX, or UNIX formats are in the list of disk formats readily available. Of course, the files may be copied from an MSDOS or other format disk over serial or parallel connections to another computer.

The quality and portability of the volumes in the library varies significantly, due to the variation in the source and purpose of the programs and due to the variation in the skill and experience of the developers and maintainers.

Following is a short list of volumes of potential interest to 6809 and 680x0 users:

```
115    Ed Ream editor for small C
126    Martz C library
127    RAP formatter
129    Citadel bulletin board
132    6809 tools
133    E screen editor
137-8  DDJ columns
145    ROFF4 formatter
146    Small C for 6800 FLEX
147    RBBS4 bulletin board
149    6800/1802 assemblers
155    B-trees, FFT, etc.
160    Programs from Learning to
            Program in C
161    Programs from Efficient C
165    Programs from Reliable
            Data Structures in C
168    Simple data base
172-3  LEX processor
174-5  YACC processor
176    XLISP processor
182    UNIX bulletin board
185    Sort utilities
190    68K assembler
197-8  MicroEMACS screen editor
```

Microware is marketing the MicroEMACS screen editor for OS-9/68K, so converting the public domain version to OS-9/68K is probably not worth the effort.

The C Users' Group may be contacted at the following address:

C Users' Group
Box 97
McPherson, KS 67460

or at the following telephone number:

316-241-1065

ROUNDOFF ERRORS

Numerical results on digital computers may be incorrect due to errors of multiple types, derived from several sources. The categories of error sources are those inherent in the mathematical

modelling of the real-world problem, those caused by radix and other conversions, those induced by modelling of the real-world problem, those caused by radix and other conversions, those induced by the truncated representation of numbers during computations, and those caused by errors in the statement of the desired computational sequence.

Numbers are usually represented in digital computers in one or more of several formats, designated in C as char, int, long, float, and double. Each of these formats has limitations on the space allocated for the representation of the numbers under their classifications, which must be considered by a user of these computers. Mainframe computers usually have special hardware to process each of these types of numbers in order to materially increase the processing speed. Mini and micro computers usually process several of the types with software.

Integer numbers have representations which are whole numbers, and may usually take on both positive and negative values, although not always. Since they have a limited amount of storage assigned to them, they always have restrictions on their values. Depending upon the internal representation of the numbers, these restrictions are usually stated as a number of decimal digits or as a number of binary digits (bits).

Floating-point numbers contain a real number and an exponent. Both the real number, often called the characteristic, and the exponent, often called the mantissa, are always signed numbers. The exponent value may represent a power of two, four, eight, ten, or other base, depending upon the implementation.

The difference between an actual value and a true value is called the error. The magnitude of the error is called the absolute error. The quotient of the error divided by the true value is called the relative error.

Rounding a number is the process of establishing an approximation to a true value which may be represented in a specified number of digits. The difference between the true value and the rounded approximate value is called the roundoff error.

The number of significant digits is usually based upon the largest power of ten which may be represented exactly, given the number of decimal digits or bits allocated to each data type, ignoring scaling factors. This number is normally trivial to determine if the data type is represented in decimal and reasonably simple to determine if the data type is represented in binary. For example, a 16-bit signed binary field can contain a range of numbers from -32767 to +32768, and thus has 4 digits of precision. A 32-bit signed binary field can contain a range of numbers from negative to positive two billion (approximately), and thus has 10 digits of precision. In making the determination of the number of significant digits for floating point representations, the exponent is ignored.

The first source of errors is that caused by simplifications and inaccuracies in the modelling of the real-world system. For example, Newtonian physics provides an excellent mathematical model for the description of the movement of physical bodies moving at "normal" relative speeds in frictionless environments. The modelling becomes less accurate for those systems involving friction and/or those systems involving extremely high speeds.

The modelling of physical systems is not the only case in which errors are made due to simplification of the real-world systems. Given the same set of historical and current data, different econometric models would predict different results of the same governmental policy or business marketing change. Given the same personal or business system, different tax preparation systems would compute different amounts of money to be paid or refunded. These differences are primarily due to the variations in the world-views of the various models, not necessarily to mistakes in the models, mirroring the world-views of the model designers.

The most subtle problems caused by simplification in modelling are those implicitly stated in the assumptions under which the model was derived. Most assumptions are stated positively, if at all, leading to the potential problem of the utilization of models in situations in which they are not really applicable, because of the lack of negatively-stated assumptions.

When evaluating mathematical models on digital computers, an additional level of modelling is required. Mathematical functions such as integration which involve an infinite number of sums or other operations may not be performed directly, but must be approximated by a finite number of discrete operations.

Unless extraordinary care is taken during the construction of programs for the evaluation of such functions, the results will be inaccurate. Close does not always count, especially when a large number of small structural errors is present.

Conversion problems may occur during the decimal/binary and binary/decimal transformations usually required to allow computers to communicate with humans, other computers, and with other devices. Some computer hardware and software systems avoid the radix conversion problems by performing all or some internal operation in decimal, rather than in binary, and some of the comments below obviously do not apply to such systems. Decimal operations are

generally slower than binary operations, and are wasteful of memory in many cases. Thus hardware and software designers tend to emphasize binary data storage and to de-emphasize decimal storage, in general.

Although integral decimal numbers may be easily converted exactly to binary, and vice versa (assuming no overflow), non-integral decimal numbers may have repeating binary equivalents (and vice versa), and thus no finite number of decimal or binary digits is always sufficient to exactly contain a non-integral binary or decimal number.

An additional problem concerns the uncertainty of the last decimal digit or last four bits during the conversion from binary to decimal or from decimal to binary, assuming a finite number of digits of precision. Since one decimal digit requires more than three and less than four (3 and 1/3) bits to represent, converting a non-integral binary number to decimal and back will, in general, cause at least the last four bits to different, although cases may be generated in which no bits would be different or in which almost all bits would be different, due to roundoff, which is discussed later in this article.

Although radix conversion errors cannot be entirely avoided, they can be controlled. For example, if one program produces results for another program to use, it would be advantageous for no radix conversions at all to be required, thus eliminating it as a source of error. Most computer languages provide means for specifying binary input/output formats, as opposed to decimal input/output formats. Even those that do not allow binary input/output often support a scientific notation which maintains the maximum number of digits of precision, regardless of the size of the number, thus minimizing the conversion error.

Rounding errors are caused by the representation of a value by an approximation which will fit within a limited amount of precision. There are several methods used to perform this process. The oldest method, simply truncates the excess digits or excess precision; this produces results with absolute value consistently smaller than the absolute true value. Another method adds + or - decimal 5 (or binary 01) to the first insignificant decimal digit or bit, depending on the sign of the number, thus rounding the nearest least significant digit, and possibly the entire number. Still another method rounds such that the last digit is even. These methods produce random roundoffs intended to minimize the average amount of roundoff error. Although pathological cases may be shown for all roundoff methods, rounding to the nearest last significant digit is generally used and usually works well.

Why is control of roundoff error so important? At first glance, it would seem that roundoff errors would be so insignificant that they would seldom materially affect calculations. However, more careful analysis shows that roundoff errors may potentially affect every operation in a lengthy calculation.

The most obvious source, and used as examples in most texts, is the roundoff error during multiplication. In general, if an n-digit number is multiplied by an m-digit number, the result has n+m digits. When this number is required to fit into fewer than n+m digits, roundoff errors may appear.

Roundoff errors may also occur during addition and subtraction, since the sum or difference of two n-digit numbers requires n+1 digits, in general, and during the scaling of smaller numbers participating in addition and subtraction operations with larger numbers.

Since division involves multiplication, addition, and subtraction, roundoff errors may be greatly magnified, especially when dividing a larger number by a number smaller than one in magnitude.

Because transcendental, logarithmic, and other mathematical functions are usually evaluated by Taylor or other series expansions, roundoff errors must be carefully controlled at every stage of a calculation to prevent them from destroying the significance of the results.

What can be done by a programmer to minimize the results of roundoff errors in calculations? Although there are no global solutions, there are good sets of guidelines to be followed.

Structure calculations to eliminate the use of floating-point operations entirely, if possible. Thus, if a program performs calculations involving dollars and cents, use long (often 32-bit) data types, convert dollars to 100 * cents on input, and convert cents to dollars and cents on output. The only remaining rounding errors may involve such tax or other calculations which then result in an integer number of cents.

Order calculations to attempt to perform operations on like-size numbers. For instance, perform the following computation with ten decimal digit floating-point accuracy:

```
 1.0000000000
+1.0000000000*(10 to -15 power)
-1.0000000000
-------------
 0.0000000000
```

Although every number entering the calculation is exact, the result is at least slightly surprising; however, if the calculation were ordered correctly, the following results would be more as expected:

```
1.0000000000
   -1.0000000000
   +1.0000000000*(10 to -15 power)
   -------------
   +1.0000000000*(10 to -15 power)
```

If specific truncation is necessary, such as in the case of tax or other monetary calculations to the nearest penny or dollar, perform it as early as possible in the operational sequence. This will minimize the common error in which the sum of the columns of a matrix disagrees significantly with the sum of the rows. Such problems are not only embarrassing; they may violate audit and governmental regulations requiring crossfoot errors to be less than one dollar.

Never use direct equality tests involving floating-point numbers. Instead, use group tests, such as "greater than or equal to" or "less than or equal to", or interval tests, such as "abs(expression) less than" or "abs(expression) greater than", etc. Avoid any use of non-integral loop counters, since such structures often loop one too few or one too many times, because of roundoff error.

Although there is no totally effective means of eliminating programming or design errors, there are means to detect errors and to hopefully correct them without creating new ones, on an iterative basis.

The most effective time to eliminate errors is in the statement of the problem. Given a correct statement of a problem, errors can occur at any stage of the design, analysis, programming, or implementation of a program or of a system. Programming texts provide many hints for checking the results of programs against standards, etc.

A tool which is so commonly used to help prevent programming errors during calculations that it is often forgotten is the standard library. Imagine the chaos which would result if every calculation were written in a low-level language such as assembler or machine language, and no common libraries were used; programming errors and calculation errors would be rampant. Even in low-level languages, standard libraries are available to perform extended precision and floating-point calculations. For example, Motorola offers the MC6839 IEEE standard floating-point package to assist 6809 users in such calculations in assembler language. Many C compilers offer one or two floating-point data types (float and double) and a library of floating-point routines.

One danger in using such tools to assist in the programming process lies in misunderstanding their limitations and strengths, such that the limitations may potentially corrupt the calculations being performed.

Following are several sample programs intended to demonstrate specific points about roundoff error, as related to the number of significant digits carried by various language implementations.

The first is a C program designed to show the differences a few digits of significance can make in extreme cases. C compilers often support two lengths of floating point numbers, limited to two fixed numbers of bits. The constant is chosen to overflow the number of significant digits of the shorter floating-point representation, but not the longer. Thus the double-precision value is slightly greater than one, but the single-precision value is equal to one, and one to any power is still one.

```
#include <stdio.h>

main()
{
     int i;
     float a;
     double amt,b,c;

     amt = 1.00000001;
     a = amt;
     i = 0;
     b = amt - a;
     c = b * 100 / amt;
     printf(" %2d %15.7f %12.7f %6.1f\n",
          i, amt, a, c);
     for (i = 1; i < 31; ++i)
     {
          amt = amt * amt;
          a = a * a;
          b = amt - a;
          c = b * 100 / amt;
           printf(" %2d %15.7f %12.7f
           %6.1f\n",
              i, amt, a, c);
     }
     exit(0);
}
```

The output produced by this program appears below. The first column represents the iteration number, the second represents the double precision value, the third represents the single precision value, and the last represents the percentage of relative error.

```
0      1.0000000    1.0000000    0.0
1      1.0000000    1.0000000    0.0
2      1.0000000    1.0000000    0.0
3      1.0000001    1.0000000    0.0
```

4	1.0000002	1.0000000	0.0
5	1.0000003	1.0000000	0.0
6	1.0000006	1.0000000	0.0
7	1.0000013	1.0000000	0.0
8	1.0000026	1.0000000	0.0
9	1.0000051	1.0000000	0.0
10	1.0000102	1.0000000	0.0
11	1.0000205	1.0000000	0.0
12	1.0000410	1.0000000	0.0
13	1.0000819	1.0000000	0.0
14	1.0001639	1.0000000	0.0
15	1.0003277	1.0000000	0.0
16	1.0006556	1.0000000	0.1
17	1.0013116	1.0000000	0.1
18	1.0026249	1.0000000	0.3
19	1.0052566	1.0000000	0.5
20	1.0105409	1.0000000	1.0
21	1.0211930	1.0000000	2.1
22	1.0428351	1.0000000	4.1
23	1.0875050	1.0000000	8.0
24	1.1826671	1.0000000	15.4
25	1.3987015	1.0000000	28.5
26	1.9563659	1.0000000	48.9
27	3.8273676	1.0000000	73.9
28	14.6487428	1.0000000	93.2
29	214.5856666	1.0000000	99.5
30	46047.0083283	1.0000000	100.0

The following example is a short C program designed to illustrate the dangers of using a non-integral value for a loop control variable. Various language processors produce different results when presented with such constructs.

```
#include <stdio.h>

main()
{
    int j;
    float i;

    for (j = 0; i = 0.999999999;
        i < 9.99999999;
        i += 0.999999999, ++j);
    printf("%d\n", j);
    exit(0);
}
```

The correct answer is 10; however, many C programs print 9 or 11.

The final example is intended to demonstrate the dangers of ignoring roundoff when using textbook solutions. Remember the formula provided in high school for determining if the roots of the polynomial (a*x*x+b*x+c) are real, equal, or imaginary? It compares the value of the expression (b*b-4*a*c) for positive, zero, or negative. However, for values near zero, the roundoff problem becomes crucial, and values of exactly zero will not often be encountered, in general. The usual solution is to assume that any value larger than some very tiny positive number is positive, that any value smaller than some very tiny negative number is negative, and that numbers falling into the intermediate range are zero. However, the coefficients a, b, and c of the polynomial could themselves each be very small, leading to potential problems unless they are scaled first.

```
#include <stdio.h>

main()
{
    float a,b,c;

    while (1)
    {
        printf("a, b, c\n");
        sscanf("%f%f%f", &a, &b, &c);
        if (!a)
            break;
        d = b * b - 4 * a * c;
        printf("%f %f %f %f %s\n",
            a, b, c, d,
            (d > 0) ? "positive" :
             (d < 0) ? "negative" :
            "zero");
    }
    exit(0);
}
```

The programmer must be aware of the problems which may be caused by roundoff and other errors encountered during numerical computations. These problems are not confined to scientific and process control classes of programs, and may affect seemingly simple calculations in game or business programs.

They must be particularly wary of the C compilers which offer less than the equivalent of 10 decimal digits of precision when performing calculations involving money; some reports submitted to governmental organizations must be correct to the penny, and others must be correct to the dollar. There may be civil penalties for violations of such regulations.

EXAMPLE C PROGRAM

Following is this month's example C program; it converts assembler equate files to C definitions.

```
#include <stdio.h>
#include <ctype.h>

char *p, *q, *r, string[256];
FILE *fd = stdin, *td = stdout;
```

```c
main(argc, argv)
int argc;
char *argv[];
{
    if (argc > 1)
    {
        if (*argv[1] == '-')
        {
            if (argv[1][1] == '?')
            {
                fputs("Usage: ", stderr);
                fputs(argv[0], stderr);
                fputs(" [input-file [output-file]]\n\n",
                    stderr);
                fputs(" where: input-file is input file name\n",
                    stderr);
                fputs("        output-file is output file name\n\n",
                    stderr);
                fputs(argv[0], stderr);
                fputs(" converts assembler equates to C code.\n\n",
                    stderr);
                fputs("Values are denoted by:       name.\n",
                    stderr);
                fputs("Addresses are denoted by:    ptr(name).\n",
                    stderr);
                fputs("Single bytes are denoted by: peek(name).\n",
                    stderr);
                fputs("Double bytes are denoted by: dpeek(name).\n",
                    stderr);
                fputc('\n', stderr);
                exit(0);
            }
        }
        else
        {
            if (!(fd = fopen(argv[1], "r")))
            {
                fprintf(stderr, "%s: cannot open %s\n",
                    argv[0], argv[1]);
                exit(1);
            }
        }
        if ((argc > 2) && (!(td = fopen(argv[2], "w"))))
        {
            fprintf(stderr, "%s: cannot open %s\n",
                argv[0], argv[2]);
            exit(2);
        }
    }

    while (fgets(string, 256, fd))
    {
        for (p = string; *p; ++p)
            if (*p == '\t')
                *p = ' ';
        for (p = string; (*p == ' '); ++p);
        switch (*p)
        {
        case ';':
        case '*':
            putc('/', td);
            putc('*', td);
            if (*++p != ' ')
                putc(' ', td);
            while (*p >= ' ')
                putc(*p++, td);
            if (*(p - 1) != ' ')
                putc(' ', td);
            putc('*', td);
            putc('/', td);
        case '\n':
        case '\r':
        case '\0':
            putc('\n', td);
            continue;
        }
        if ((!isalpha(*p)) && (*p != '_'))
            continue;
        for (q = p; (*q > ' '); ++q);
        while (*q == ' ')
            ++q;
        if ((*q != '=') &&
            strncmp(q, "equ ", 4) &&
            strncmp(q, "EQU ", 4))
            continue;
        while (*q > ' ')
            ++q;
        while (*q == ' ')
            ++q;
        if (*q < ' ')
            continue;
        fputs ("#define ", td);
        while (*p > ' ')
            putc(*p++, td);
        putc(' ', td);
        putc('(', td);
        while (*q > ' ')
        {
            if (*q == '$')
            {
                putc('0', td);
                putc('x', td);
                ++q;
            }
            else
                putc(*q++, td);
        }
        putc(')', td);
        putc('\n', td);
    }

    fputs ("\n#ifndef peek\n", td);
    fputs ("#define peek(x) *((char *)(x))\n", td);
    fputs ("#define dpeek(x) *((int *)(x))\n", td);
    fputs ("#define ptr(x) ((char *)(x))\n", td);
    fputs ("#endif\n", td);
    if (fd != stdin)
        fclose(fd);
    if (td != stdout)
        fclose(td);
    exit(0);
}
```

EOF

Basically OS-9

Dedicated to the serious OS-9 user.
The fastest growing users group world-wide!

6809 - 68030

A Tutorial Series

By: Ron Voigts
2024 Baldwin Court
Glendale Heights, IL

MACROS ANYONE?

The word MACRO is Greek in origin. It means large. If you looked it up in the dictionary, it would tell you something like: a prefix meaning large, long, great or excessive. Macro is also a part of computer jargon. It also prefixes a word. The more formal name is MACROINSTRUCTION. If you can create a group of instructions and execute them with one instruction, then you have created a macroinstruction. Or macro for short.

Sometime back I wrote a column on the OS-9 procedure. It is a file that contains lines of OS-9 instructions. They can be executed by the shell. Or they may invoke another procedure file or program. A prime example is the STARTUP file. Usually it contains SETIME and other things that you may want to have occur at start up time. You don't need to type all those lines. Just enter the file's name.

A more basic form of the macro occurs at the computer architecture level. At the gut level of the microprocessor are sets of instructions. These are even more primitive than the ones found in writing assembly language programs. One micro-P that I studied had only 16 instructions. And two of them were No Operations (NOP)code. From these the standard set of computer op codes are created. It is easiest to examine one of these. Let us say we write the following in our assembly language program:

```
ADDA  $25
```

The sequence of code to effect this would be:

```
IR <-- M[PC], PC <-- PC+1
TR <-- M[PC], PC <-- PC+1
A <-- A + TR
```

The IR is the Instruction Register. M is memory location. PC is the program counter. And TR a temporary register. This sequence says to move the contents of memory location pointed to by the Program Counter into the Instruction Register. The Program counter is incremented by one. The next memory contents, the $25, is moved to the Temporary Register. Again the program counter is incremented. Finally, the two registers are added and stored in Register A. This all occurs in a few machine cycles. But the programmer does not have to worry about the instructions above, he uses the macro ADDA and everything is done automatically.

This is a simple, basic example of the macro. Everything is relative. It turns out the sequence of events are really called microinstructions. The op code is simply an instruction. But it really is a macro that causes a number of cycles to occur inside of that little black chip called the CPU.

MACRO ASSEMBLERS

At the assembly language level is the macro assembler. This is an assembler that permits us to create macros while working in assembly language. Microware has a rather nice one called the Relocating Macro Assembler. This is the same one provided with their C language compiler. With it you can create instruction sets that can be called with a psuedo opcode.

Let us say periodically you want to clear the D register in the 6809. There is no one code for it. But you could create the following:

```
CLRD    MACRO
 12
CLRA
CLRB
ENDM
```

Now, anytime you wanted D to be cleared just use CLRD. At assembly, the CLRA and CLRB will be substituted for the psuedo opcode.

There are three parts to creating this macro. First is the header. This is its name with MACRO after it. Next is the body of the macro. Finally, it is ended with the terminator. ENDM indicates the macro end. Arguments can be passed to the macro, too. These are indicated with the back slash and a number. Then can be \1 through \9. Here is a simple macro:

```
PRINT MACRO
 11
LEAX \1,PCR     GET ADDRESS OF STRING
LDA  #\2        GET OUTPUT PATH
LDY  #\3        GET STRING LENGTH
OS9  I$WRITE    WRITE IT!
ENDM
```

Anytime we want to print something, just enter:

```
PRINT STRING,$01,LENGTH
```

At assembly time this one line will become:

```
LEAX STRING,PCR
LDA  #$01
LDY  #LENGTH
OS9  I$WRITE
```

One simple line of code becomes 4 program lines. Every time the PRINT macro is used 4 lines are generated for it.

There is some confusion between a subroutine and a macro. Hopefully, this can help to settle it. It is easy to see a parallel between the two. Both receive parameters and both accomplish some function. The big difference is that one is a separate routine, while the other is a templet for a function. When the macro is used, the lines of instruction are generated in the program. Each time the macro is called, it generates the same lines of code. On the other hand, the subroutine is an actual routine that is entered. It can be entered from anywhere with the return address pushed on the stack. At its end, a RTS puts the return address back in the Program Counter and execution continues from the calling routine. So the subroutine exists in one location. It is entered and exited. The macro recreates the same routine every time it is called. The subroutine is created only once.

With regards to this, care must be taken when creating labels with the the macro. Remember each time the macro is called, it gets "rubber stamped". If this is done more than once and it has some internal labels, there will be multiply defined labels. To overcome this dilemma, the RMA permits you to create unique labels for each created macro. A label is specified with:

```
\@?
```

The question mark is what you add. At compile time the label is expanded to:

```
@xxx?
```

where the three x's become a unique number from 000 to 999. Say you declared the label in your macro as:

```
\@ABC
```

The macro would generate @000ABC for the first time, @001ABC the second time and so on. This gives you about 1000 times that the macro could be called.

Two other handy items are:

```
\Ln  and  \#
```

The first one returns the character length of the n argument. The second returns the number of arguments passed. In the example given before using the macro PRINT, it might be advantages to know if 3 arguments are passed. At compile time, we want to know if an error has occurred in specifying the macro PRINT. After the first line we can add:

```
ifne \#-3
FAIL PRINT: Must have three arguments
endc
```

Now should we try to compile the source code and declare the incorrect number of parameters, the error counter will be incremented and our fail message displayed. The \Ln works in much the same way. It can check for incorrect argument lengths.

EDIT MACROS

Many word processors and editors allow macros to be created. A writer friend reports of a word processor that permits defining some keys to generate groups of characters. Ideally the writer encodes certain frequently used words into the macro key table. They may include names, often used phrases, and whatever else suits the writer's imagination. Then instead of typing out the word, the proper key is pressed. Poof! The word appears almost instantly.

The standard OS-9 Editor has a facility to make macros. Macro commands can be made that combine edit commands. Complex edit sequences can be accomplished with a single line entry. Newly created macros can be saved to disk. And older ones can be reloaded. The macro consists of two parts. There is a header which contains the macro name and a variable list. And there is the body of the macro. This contains the edit commands that produce the desired results.

The header name can be of any size, although a short name is probably more desirable than a long one. After all, the macro is used to save work, and who wants to enter long names? The name is not sensitive to case. So either upper or lower case can be used. The name is followed by the parameter list. Two types of arguments are permitted. They are string and numeric. If a variable name is preceded by a #, then it is numeric. If a $ is there, then it is string. By way of example, let's say the first line of the macro is:

```
FIND #N $STRING
```

When this macro is called from the editor, it could be entered as:

```
E:.FIND 12 "Hello"
```

The dot before the macro name indicates that FIND is indeed a macro. In the call to this macro, #N would be equal to 12 and $STRING would be "Hello". Parameters must be passed in proper order. Once in the main body of the macro, they are referenced by their variable name, just as a program would do.

As said earlier, the main body of the macro consists of commands that could have been entered in response to the editor prompt. Any series of commands, entered while in an editor session, can be part of a macro. There are some items that appear more often in a macro. First, there is the brackets and semicolon. There syntax is:

```
[ ... ] n
```

What appears in the bracket is executed for n times. If something fails within the brackets, they will be exited prematurely, before n counts have occurred. If a colon is added into the loop a conditional exists. Its syntax appears:

```
[ ... : ... ] n
```

As long as the editors internal fail flag is clear, everything in front of the colon will execute. As soon as an operation causes it to be set, execution goes to the statements after the semicolon.

The best way to understand a macro is to create one and examine it. To start a macro we can enter:

```
E:.MAC //
```

This says to edit a macro that is new. If a name were placed between the delimiters that particular macro would be edited, as long as it existed. Now we can create a macro that finds the Nth occurrence of a string and deletes the line it is in. So we enter the following

```
M: DELLINE $S
M: ! THIS MACRO DELETES ALL LINES
M: ! THAT HAVE $S IN IT.
M: "                 ! MOVE TO FIRST LINE
M: [                 ! LOOP 1
M: .NEOB             ! TEST FOR END OF BUFFER
M: [                 ! LOOP 2
M: .STR $S  D        ! DELETE IF $S IS PRESENT
M: :                 ! ELSE
M: +                 ! ADVANCE TO NEXT LINE
M: ]                 ! END OF LOOP 2
M: ] *               ! END OF LOOP 1
M:Q                  ! QUIT
```

Lines that are part of the macro are indented one space, just like it is done while using the editor. Most of the standard edit commands can be used while entering a macro. The working of this macro is simple. On entry $S is the target string. We move to the start of the buffer. Loop 1 test for the end of buffer with .NEOB. As long as this is true (the fail flag is clear), the contents of loop 1 are executed. Loop 2 tests for the presence of $S on the current line. .STR clears the fail flag if the string is found, otherwise it is set. As long as the line has $S in it, it is deleted. Now, if $S is not found the fail flag is set and execution occurs after the colon. This merely advances to the next line. In short, this macro test a line for a string. Delete it if present, else advance to the next line and try again.

This is a trivial case. It may not be very practical or useful, unless you are trying to purge a file of some unwanted record. You can create more complicated and useful ones, then I have shown here. When you have something worthwhile saving just enter:

```
.SAVE "MACRO1 MACRO2"MYMACROS"
```

Here two macros are saved to a file called MYMACROS. Later when entering a new edit session, enter:

```
.LOAD "MYMACROS"
```

and they are back in the macro area of the editor. You can also enter:

```
.DIR
```

and you will see a listing of the buffers and macros that you have.

There is a very useful set of commands that set the fail flag if the condition is not true. These give you the means to execute loops and commands.They are in brief:

```
.EOF        at end of file
.NEOF       not at end of file
.EOB        at end of buffer
.NEOB       not at end of buffer
.EOL        at end of line
.NEOL       not end of line
.ZERO n     is n zero
.STAR n     is n 65535 (*)
.STR str    is str in line
.NSTR str   is str not in line
.S          exit loop and clear flag
.F          exit loop and set flag
```

If you do a lot of work with the OS-9 editor, you'll definitely want to make use of macros. The offer a way to do complex repetitive tasks quickly and easily.

MACROS IN C LANGUAGE AND AN ANSWER TO A PUZZLE

C language has its version of the macro. It is called the macro substitution. If you have done any programming in C, you will quickly

recognize the preprocessor command #define. They are usually at the start of the program and take a form something like:

```
#define CLEAR 12
```

This tells the compiler to substitute 12 for CLEAR in the source code prior to compiling.

A macro can also be defined with arguments. A simple statement can be created that will be replaced with another that has been previously defined. Let's look at an example.

```
#define cube(x) (x*x*x)
```

Now later when the program is executing, we enter a line:

```
y=cube(s)
```

But during the preprocessor phase the line becomes rewritten:

```
y=(s*s*s)
```

Notice that the variable used is different from what is declared in the #define statement. In fact, nothing has been said about the variable type. The substitution occurs. Whether it can be compiled is up to you.

Say you have a function called swap. It takes two integers and swaps them. This function could be written:

```
/* Swap two integers */
int swap(i,j)
int *i, *j;
{
  int t;
  t=*i;
  *i=*j;
  *j=t;
}
```

This could be replaced with the following.

```
#define swap(i,j) t=i; \
                  i=j; \
                  j=t;
int t; /* temporary storage */
```

They will both do the same thing. What is the difference? The first is a function. It is a separate routine. Each time it is called, the arguments' address pointers are pushed on the stack. Using their pointer values in swap(), they are swapped around using t as a temporary location. The second method is a macro substitution. Each time a swap occurs the token is replaced by the program lines. No calls are made to outside routines. And pointers don't have to be used. By the way, those reverse slashes indicate continuation of the code on the next line. If you use the Microware C Compiler, than you would have to put the code on the same line. So the line would appear:

```
#define swap(a,b) t=i; i=j; j=t;
```

Either way though the result will be the same.

Occasionally code does not turn out as you may have intended. Last month I left you with a programming mystery. I use a C define macro that worked fine, but ended in generating a longer object file. The executable module was longer then it needed to be. I had attempted to write a macro that would print out strings. It was:

```
#define OUT 1
#define print(s) write(OUT, s, strlen(s))
```

Using it would make things much easier. To write out the string "HELLO", all I needed to type was:

```
print("HELLO")
```

Pretty easy, eh? Well it did what it was suppose to do. But during the precompiler pass the line would be changed to:

```
write(1, "HELLO", strlen("HELLO"))
```

What do you think the assembly code looks like after the C compiler has converted it? Here is my annotated version of it:

```
* *   print("HELLO");
* Get the string length of "HELLO"
* and save it on the stack
leax _2,pcr
pshs x
lbsr strlen
leas 2,s
pshs d
* Save the location of "HELLO"
leax _1,pcr
pshs x
* Save the standard output path
ldd #1
pshs d
* Branch to WRITE
lbsr write
leas 6,s
* Printed string "HELLO"
_1
fcc "HELLO"
fcb $0
* String checked for length
_2
fcc "HELLO"
fcb $0
```

The string length of "HELLO" is first checked with strlen(). The result is returned and saved on the stack. Next the location of "HELLO" is pushed on the stack. Notice that it

is not the same "HELLO", whose string length was checked. Finally, the standard output path is pushed and a long branch is made to the library routine WRITE.

The important thing here is that our string gets repeated twice. C has no way of knowing that it is the same. As far as it is concerned, the strings are two different entities. Now take this effect and reproduce it over and over again, and you've got a program that is much longer then it should be. Every printed text line gets repeated twice in the executable file. So what do we do?

First, we could correct the assembly code created by the C compiler. This would mean going through the entire assembly code. All double entries would have to be erased and references to them adjusted properly. In the previous example, the label _2 and the two lines after it would be erased. Then, the references to _2 would be changed to _1. This would need to be done for each occurrence. Last months program contained many needed corrections. The job could become a little tedious and possibly prone to mistakes. So, it might be advisable to find another solution.

Another way to correct the problem is to assign the string to some pointer. Say we declared a pointer with "char *t". Then every time we wanted to print something we would set t to point to it. Our previous example would be rewritten:

```
t = "HELLO";
print(t);
```

This method is not really very clean and neat. It also defeats the entire purpose of having the macro. We are trying to eliminate work. Not create it! And we are trying to make a simple, but better string print routine.

Finally, we may have to look into changing print() from a macro to a function. This may be the best solution. It does not require rewriting the assembly code. It means no extra program rewrite. And it is the simplest method to correct the problem. So, I deleted the macro line. And added the following function.

```
print(s)
char *s;
{
    write(OUT,s,strlen(s));
}
```

Remember the difference between macro and subroutines! This becomes a totally separate part of the program. The pointer to the string is passed to it. In a sense this is the solution I outlined second. A pointer is used as the argument instead of the string. But we only had to add this simple 5 lines to the program. No major changes were made.

A FEW CLOSING THOUGHTS

The idea of replacing a list of instructions with a one line input is common to many systems. In some cases complex instructions are replaced with a few keystrokes. The whole idea is to make life easier. After all who wants to do any more work than is necessary? If you are into assembly language programming you'll find assembly macros save retyping repetitive routines. The same is true for C macros. If you use the editor frequently, I am sure creating macros will save time and speed up programming. And if you're like me, you'll create procedure files to carry out long repetitive jobs.

I like to consider what future advancements could be made. Some years back I had a program that allowed macro keys to be defined. This was before my days with OS-9. The number keys when pressed with the control key could be defined to print out commonly used commands. Holding down the control key with a number caused an entire command to appear. Maybe a new OS-9 shell could be created that permitted such a thing. Imagine a ^1 and the current directory would appear. Perhaps a ^2 and the date and time would be there. The list could go on.

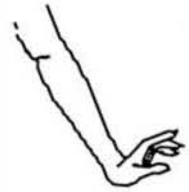
As I pointed out before, I like to think of procedure files as macro lists. Type the files name and everything in it executes just as if you entered it from the keyboard. How about a procedure oriented language? Create procedure files that can handle conditional statements, parse the input line for parameters and allow simple variables to be created. You'll have procedures that don't just execute a list of commands, but can be flexible and make decisions during execution.

How about a pre-processor for BASIC09? Write a program with Basic09 macros. Run the pre-processor on it and convert the macros to real Basic09 statements. This would sure cut down writing little routines to handle simple items. And it would reduce the clutter in the directory.

These are a few thoughts of mine. There is so much that can be done. But, even if you not the inventive type, use what is available. They can make things so much easier. Above all, have fun!

EOF

Terminal Wrist



SOFTWARE USER NOTES

A Tutorial Series

By: Ronald W. Anderson
3540 Sturbridge Court
Ann Arbor, MI 48105

From Basic Assembler to HLL's

You've heard of Tennis Elbow? I have a new ailment that I call Terminal Wrist. Is anyone else out there nuts enough to sit at a terminal all day at work and then go home and do the same thing? Years ago, I stopped holding my hands poised above the keys as I used to do with a manual typewriter, and yielded to the sloppy programmer practice of resting the heel of my hand on the front edge of the keyboard while typing. If the terminal is on a table just at the proper distance, my wrists end up laying right on the edge of the table. However, my palms always lay on the edge of something and support the weight of my arms. When I really get involved the heels of my hands get sore and stay that way until I take a day or two away from the terminal.

I had an additional irritation on one of my terminals. It seems the whole thing was made with a very rough textured finish, though it is molded plastic. It felt like resting my hands on sandpaper! I fixed it by using some very fine sandpaper to take the roughness away. The improvement is most welcome. I suspect we all tend to work with the terminal too high or our chair too low. You really shouldn't be reaching up for the keyboard, and that is probably at least part of the cause of the discomfort. Anyone have any good ideas? I've lately been thinking about a piece of carpeting under the keyboard and extending over the table edge.

PLuS Again

For those of you who are new to this publication and to computing, PLuS is a computer language very much like Pascal only better for some applications, particularly those leading to operation of the program from ROM in a dedicated computer application such as a machine control or electronic measuring instrument. PLuS comes from a language written for the 6809 several years ago by Graham Trott in England for Windrush Micro Systems Ltd. PLuS offers several advantages, particularly to those who have programmed in PL/9. The new compiler is very similar to PL/9 so that program "translation" is almost trivial. It is an "interactive" compiler in that during compilation, an error will stop compilation and allow you to re-enter the edit mode and fix the error. Then you can restart the compilation.

"interactive" compiler in that during compilation, an error will stop compilation and allow you to re-enter the edit mode and fix the error. Then you can restart the compilation.

I've heard from Windrush again just recently, with a newer version of PLuS than I had a month ago to begin testing. I managed to get a non-trivial program to run on the Mustang, and the standard speed improvement factor of 7 over a 2 MHz 6809 was duels noted. The PL/9 version is 5577 bytes and the PLuS version is 6862 bytes. When you consider that the average instruction for the 68000 is considerably longer than the average instruction for the 6809, the small increase in size is impressive. The speed is certainly impressive, as is the speed of the PLuS compiler running on the 68020. Windrush is to be complimented on a couple of little improvements over the 6809 version. First of all, comments may now be nested. That means that one can comment out a whole section of program without getting into trouble for having commented out some comments, a real bother in the 6809 version.

I shipped off a bug report via international express mail and an update arrived just 11 days later by the same mail service. I've tested it and found that the bugs I had reported were completely eliminated, though a few more have been found, but they are just about in the trivial category, for example, the compiler doesn't report an error in the last line of the program!

I thought I would port over my FFT program that was in last August Micro Journal and found that in the process of getting it to run under PLuS, another bug came to the surface. It was hard to pin down, but easy to program around once found. Another bug report is off to Windrush. At that point, I converted several more PL/9 programs to PLuS and uncovered no more bugs, so I think they are rapidly getting down to the point of having a very good compiler.

I noted as I went through the programs that there are a couple more differences between it and PL/9 than I had first noticed. First of all, there is no built-in square root function. I coded one in PLuS and added it to the scipack.lib. I found after some optimization that I could get 10000 square roots in 4.64 seconds, or 464 microseconds per square root, and that was in a loop in which the loop indexing surely took a non negligible part of that time. The same test program using the built-in square root function of PL/9 on a 2 MHz 6809 system took how many times longer? You guessed it, the 6809 did 10000 square roots in 30 seconds, about 6.5 times longer!

Another new feature is some added flexibility in the case statement. PLuS allows multiple cases to cause the same action, and it even allows groups of cases more or less like the IN statement of Pascal.

```
if char
    case 'a, 'c then do-something;
    case 'A-'Z then uppercase = true;
    case 'A-'Z, 'a-'z then alfa = true;
    case '1-'9 then numeric = true;
    case 0-31 then control = true;
```

The PL/9 editor is a line oriented one, and I've always thought it to be quite handy, being much like the old TSC EDIT. The PLuS editor is screen oriented and the control keys are completely configurable by the user via the SETPLUS utility. It allows you to move the cursor in any direction, goto a line (by number) find a string, replace a string instance by instance or globally, and overall is VERY handy. When done editing, you invoke the compiler. If the compiler detects an error you can hit return and find yourself back in the editor with the cursor right at the error in the source file. Mighty handy for quick fixes of syntax errors in the program. There is a compile only option in which you do nothing but check to see that there are no compile errors. You can compile to an output file with a listing to the terminal, and many other options.

I had written a couple of simple utilities in 'C' for the Mustang. One was to initialize the Televideo terminal (to get rid of the stupid status line at the bottom of the screen and set the cursor to non blinking). The other was to cause my printer to page (just send it a formfeed). In 'C' these were both about 2250 bytes of object code. I wrote them in PLuS, and found that I could do them in about 250 bytes each. I am not saying that PLuS is ten times more efficient at generating output code than "C", but because the library files are included at the source level, the user can eliminate all the unused parts of the library and really trim the program down. For example, in the case of the terminal initialization, all I needed from IOSUBS.LIB was the PUTCHAR procedure, and that called the i_write procedure in the OS9.LIB file. I simply edited both libraries into my program (rather than using the include function) and deleted all the things I didn't need. The PAGE utility needed the open and close file procedures as well in order to open the path to the printer and output the formfeed to it. I guess what I am really saying is that the user has a little more control over his program with PLuS, of course at the expense of having to spend a little time optimizing the program.

At this point, with 20 Mbyte hard disks and 2 Mbyte memories, surely someone will write and ask what difference it makes if a dumb utility program occupies 200 bytes or 2K. At the other end of the spectrum someone will write and say that my simple utility can be done in 60 bytes of 68000 assembler code.

Of course I expect to keep you updated on this compiler as it progresses. Let me just say that for me, at least, it will help ease the transition to the 68XXX processors in my work and my hobby activities.

HELP

I have had reports of a bug in PAT in moving a block of text within a file. Though I haven't had the problem personally, even my co-worker who uses the same computer as I, has reported it to me. Apparently the wrong area of memory gets moved sometimes. If any of the PAT users who read this can tell me how to make it happen consistently, I think I can cure it quickly. I have known about this problem for a long time and have thought that I had cured it several times. I think it was introduced when I did some cleanup of what happens at the end of a text file. There is also another bug that occurs sometimes when a file is saved. Several blank lines are saved at the end of the file. I've had that happen to me several times but I can't seem to catch what causes it. I have tried leaving the cursor several lines below the end of the file on exit, but that doesn't seem to do it.

If any of you know how to cause either one of these problems to occur, and can give me the sequence of actions leading up to their occurrence, I'd really appreciate the information.

MORE ON STANDARDIZATION

I just received Terry Ritter's permission to quote his letter to me that I mentioned last time. He was agreeing with me that standardization is the key to success in the computer business, at least from the standpoint of the computer and computer component manufacturers. He points out that I could buy a 20 Mbyte hard disk and interface for the IBM compatibles for much less than I paid for a pair of 8 inch floppys and interface for the 6809 system. Of course much of that price reduction is due to the large volume of such units that are now produced. That's the point. The standard IBM is out there in such numbers that the quantities have caused competition among the various disk drive suppliers. They have been able to simplify their designs and reduce costs, or rather they have been forced to do so in order to remain competitive.

Terry goes farther and in retrospect, indicates that the most probable reason for the failure of the 6809 was the lack of on chip memory expansion. Certainly there were chips available to do that function, but everyone who built 6809 hardware did it his own way and ignored everyone else. Not only did the software suppliers have to allow for all the various terminals to work with their software, they had to do a different version for each computer manufacturer's product, further increasing the programming effort required to cover a relatively tiny market.

A Comment

I hope I am not getting too repetitious with this theme, but I have to point out that there has been little new software for the 6809 for the past couple of years. As discussed above, the market started out too small, was fragmented by lack of standardization, and now is shrinking because many of the original 6809 system users have gone on to IBM type systems or to 68000 systems.

I note just from looking around that OS-9 68K has a very large line of software available for it. Just to mention a few, we have Stylo, the screen editor that has been popular in 6809 version for many years. I guess I can mention PAT and JUST of mine. Then moving into the compilers, we have Microware "C", OmegaSoft Pascal, Lucidata Pascal, BASIC09, a Fortran, and PLuS. There are probably several others with which I am not familiar. I hear rumors of an OS-9 version of AutoCad and some other presently MS-DOS and UNIX running software. It would appear that if we have been waiting for the 68000 to get off the ground, it has.

This discussion wouldn't be complete without mention of Peter Stark's SK*DOS which seems to be coming along nicely for those of us who don't want the complexities of a multi-user operating system (nor its price). My only reservation in this area is the availability of sufficient software to make it a workable system.

OmegaSoft Pascal

My involvement with OmegaSoft and its proprietor Bob Reimiller date back several years also. We purchased one of the first releases of his "Industrial Strength" Pascal and worked with him through a number of releases and bug cures. I have always been impressed with Bob's very quick response to bug reports. We eventually completed a very large machine control system using that Pascal in 6809 version, and got to know Bob pretty well over the course of the project.

When the 68000 version for OS-9 came along, Bob called me and asked if I would test it for him. I have to say that I found only a few bugs in the early version and Bob has updated me with the very latest.

Though Pascal is noted for its abstractness, (I mean that in its "purest" form it is primarily useful for learning how to program, and for applications involving calculations or data processing), this implementation is very useful for writing programs that are to be used within a computer system and also for stand-alone computer applications in control systems. The compiler is very fast in the 68020 version, and it is written to link in only the runtime library routines that are used in a given program. The user may write his own I/O routines and substitute them for those that link to the operating system. Therefore, the output of this compiler is very flexible.

Though the compiler is multi-pass, producing an intermediate assembler source code, Bob has made it very easy to use by supplying a number of little utility programs that generate the command files to run the assembler and linking loader. This compiler allows modular compilation. That is, a large program may be broken down into modules that may be compiled separately and linked together in the final step to produce the object code. A program change in one module then does not require recompilation of the entire program, a very nice time-saving feature, though less so than with the old 6809 version because the compiler runs so fast on the 68020 system. This is a compiler that may be run immediately and easily to compile standard Pascal programs. However, there is great depth here, and a user that needs to do something special can expand his capabilities with the compiler by writing his own special I/O or other procedures.

Assembler code can be linked to the compiler and whole procedures can be written in assembler. The compiler package consists of the compiler, a relocatable assembler, and a linking loader. Also included is a "librarian" that helps manage user supplied library functions. As I mentioned before, there are several utilities included to help the user. The one that will be used most often is the LC utility (Linkage Calculator). It prompts for information about the program to be compiled, and generates a shell command file that can do most of the work for you when you compile programs. The compilation can be tailored to your method of working, compiler options, assembler options, loader options, etc. All in all, this is a very complete system.

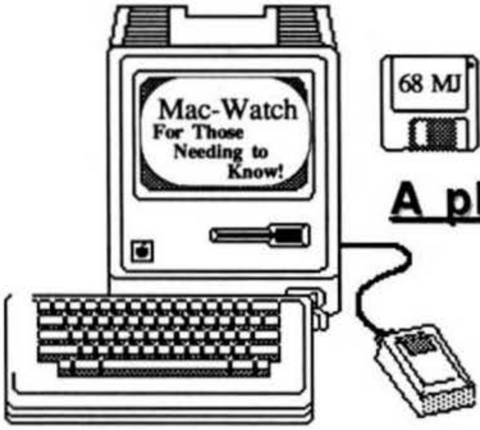
The compiler has double precision floating point math as well as single precision, so you have your choice of fast 6+ digit arithmetic or precise 15 digit arithmetic. The double precision math uses a 56 bit mantissa and one byte for exponent. 56 bit arithmetic theoretically has about 17 digit precision, but a string of calculations tend to produce some rounding errors in the last digit or so (with ANY math package). The scientific functions are good to at least 15 digits. Of course Pascal has a format specification that allows you to specify the number of digits printed. While a real Pascal purist might turn his nose up at the number of extensions to the language, it should be pointed out that those extensions don't have to be used. On the other hand, a knowledgeable programmer can make good use of HEX variables and literal constant type specifiers to make the program more understandable at the source level and to make it run more efficiently as well. Someone has described "C" as "Pascal with its sleeves rolled up" or as a Pascal that is "not afraid to get its hands dirty." This Pascal implementation fits those two descriptions very nicely. It has all the features necessary for it to get in and get the job done, be it an application that runs in a substantial computer or one that will run in stand alone hardware in a machine control or smart instrument.

OmegaSoft Pascal comes with the source code to the entire runtime package so the user can get in and tune or add things for his own applications. I can only say that the company has been around for a number of years, has provided excellent service, has upgraded the product several times, and is friendly and easy to deal with.

Late Notes

We (Hines Industries) have received our Peripheral Technology 68008 Single Board Computer. We have both OS9 and SK*DOS for it and we will be getting up to speed on it with both OS's. We have a couple of floppys and a hard disk to install on it as soon as we have the time and can get our hands on a power supply for it. I'll have a great deal more to report here in the near future.

EOF



The Macintosh™ Section

Reserved as a

A place for your thoughts

And ours.......

Mac-Watch

An Easy 512k Mac Memory Upgrade

By:
Joe Britt

Not many people still have 128k Macs, but those still do may not realize just how simple the upgrade path to 512k is. No, I don't mean a dealer mombo-swap upgrade, I mean either you or a technically-oriented friend doing it TO your 128k system.

First off--you must have no qualms about opening up your Mac and fiddling about inside. This upgrade WILL invalidate your warranty (but if its a 128k Mac, your warranty is probably already gone--unless you have Applecare or some similar extension). I take NO responsibility for any damages incurred while attempting or using this upgrade, nor do I guarantee it will be compatible with future systems, enhancements, or software. I personally use this upgrade on a Mac I put together, and have had no problems due to it.

Now, with that scary bit aside, let's dig in. First, a parts list:

16	256k DRAMS (200ns or faster)
16	16-pin soldertail sockets
1	74F253 or 74AS253 multiplexer
2	2.2k resistors
1	47 ohm resistor
1	1k resistor
1	glass .1uF capacitor (maybe-depends on your mombo) and a maybe a socket for the 74F or 74AS253, depending on your mombo

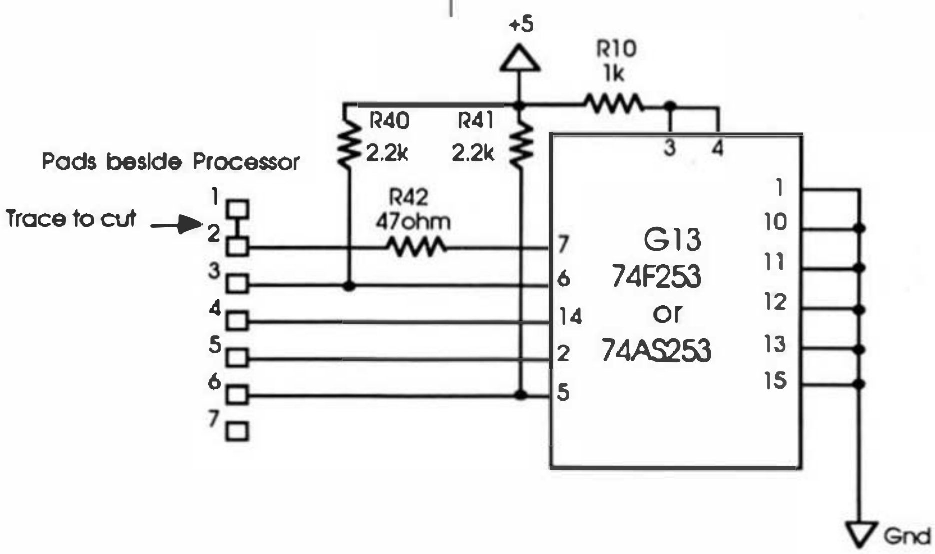
Now you need to determine what kind of motherboard you have. If you have never opened your Mac before, you are in for a treat. First, unplug and move away all external cables, peripherals, etc. If you have programmer's switch snapped on the side, pop it off with a small screwdriver. Now lay the computer (monitor-down) on a smooth, soft surface (a towel on a table is good) and remove the 5 Torx screws that hold the case together with a Torx T-15 screwdriver. Two are toward the bottom, two under the handle, and one is inside the battery compartment. The back and front are a really tight fit, but just keep tugging until the back pops off. Set the back aside. Looking down into the machine now, remove the two cables which go the motherboard. The mombo should just slide up and out now. Move the front of the case aside now, and place the motherboard in front of you, remembering static precautions.

The first (and most tedious) step is changing the DRAMS. You will recognize them because they are all together in two rows of eight, surrounded by a white silk-screened box. The best way to do remove them (without a desoldering station) seems to be this: Using a pair of very fine, sharp diagonals, clip out the 4164's, cutting the pins as close to the chip body as you can. Just throw away the old 4164's. At first this may seem like a waste, but it's well worth it. You could easily destroy your board trying to manually desolder each of those chips (not to mention destroying half of them from the soldering iron's heat). Once you have them chopped out, lift out the pin remnants with the tip of the iron. Finally, remove the solder left with solder-wick, being careful not to apply too much heat--you can easily lift traces off the board. Now, install each of the sockets and solder them in. Finally, plug in the 41256's, carefully noting orientation. If you want, you can partially reassemble the Mac and power it up now, just to verify that you didn't mess anything up. It should come back up as a 128k system as usual. If something is wrong, you will get a "sad Mac" and an error code. If the first two digits are 02 through 05, then it is a memory error. the next four digits are a 16-bit hex number that tells which bit was at fault. If you get an error, check for broken traces or solder bridges.

Next, look at your motherboard and see if you have a place that an IC could go at location G13. If you do, then you have one of the newer boards and are lucky. Just mount a socket there and plug in the chip. Since the boards are wave soldered, the holes will probably be filled with solder. Use solder-wick to remove it. Mount a 2.2k resistor at locations R40 and R41, and the 47 ohm one at R42, and the 1k one at R10. Mount the glass cap at C51. Now cut the jumper labeled "W1". You now have a 512k Mac! Reassemble it and test it out.

If you don't have an IC location at G13, then you have one of the original motherboards. All we have to do is duplicate the circuitry that goes to G13. OK, take the multiplexer and bend all the pins straight out except for 2, 8, 14, and 16. Piggyback it on the '253 at location F3 and solder those pins, noting orientation. You will see a row of 7 pads at one end of the 68000. Follow the schematic below for connection to these pads. Finally, cut the trace that connects pads 1 and 2. Reassemble your system and check it out--you should have a 512k system now.

EOF



FORTH

A Tutorial Series

By: R. D. Lurie
9 Linda Street
Leominister, MA 01543

STACKS IN FORTH

When I originally started to write on this subject, I had planned to show that stacks were easy to understand and should be easy for any one to learn to use properly. But when I actually started describing stacks, I realized that they were really not that easy to understand without some help. I had as much trouble as the next guy understanding FORTH when I was a beginner, so I thought that I should try to explain the operation of the two primary FORTH stacks in pretty much the way that I learned.

Like a lot of other people, I first encountered FORTH in the famous August, 1980, special issue of "BYTE". The collection of articles really whetted my appetite for the language, so I ordered the fig-FORTH for the 6800, along with the infamous installation guide, from FIG. After a minimum of agony, I finally finished keying in the code and got the FORTH to run, after a fashion. Since, at that time, I only had 16K of RAM, I was able to hold only a few screens in "virtual memory", but it was enough to cause a permanent addiction to FORTH!

At first, I just copied some screens in order to get the system to run, and to try to understand what was happening. My first confusion was over the Data Stack, also known as the Parameter Stack. I had to try a lot of experiments with short programs, and I even wrote a crude definition to get a simple stack picture. After a while, the concept of the stack operation finally sank in; remember, at that time, there were no books available on FORTH programming, so I had very little outside help. Mostly, I think that my own stubbornness was the deciding factor in learning to use FORTH. In any case, I hope to make learning a little easier for today's beginners.

I won't bore you with a recitation on how much easier it is now for a beginner, but will get right to the point.

The Data Stack

In order to use FORTH with any ease, you must learn how to use the Data Stack. The Data Stack is not unique to FORTH, but is used in other structured languages, such as C and Pascal. It is used to pass parameters to functions and procedures; that's what those words are doing in parentheses at the beginning of a function. The difference between the Data Stack in FORTH and in C or Pascal is that you are encouraged to manipulate the data in FORTH, but not allowed to touch the data in C or Pascal, once it is on the stack. That's one of the things I like about FORTH.

The reason for using the Data Stack, instead of a variable list, is for economy of RAM and speed of execution. Actually, once you become comfortable with the stack, you will find separate variables to be a nuisance, though sometimes indispensible.

FORTH will support variables, just as any other language does. In fact, you can use variables in FORTH in much the same way as you do in BASIC, since all variables are automatically global in nature. You can use variables to make your programs work while you are learning to use the Data Stack; that's what I did.

The most common analogy for stack operation is to compare the stack to a stack of serving trays in a cafeteria. Only the top tray is available for use, and none below can be used until the trays above it have been removed from the stack. Therefore, you don't care how many trays are on the stack, as long as there is one for you, but you can't have a tray if there are none on the stack. Trying to take a tray that isn't there is known as "stack underflow".

If you are using FF9 as your FORTH (or one with similar capabilities), you can use DEBUG as a very effective tutor into the operation of the Data Stack. Just pick an appropriate word for examination with DEBUG and watch the Data Stack grow and shrink as each operation takes place. In fact, I cannot think of a better way to learn just how the Data Stack operates.

I can show a greatly simplified example of the Data Stack in action in Figure 1. Suppose that you have written an application which requires that you calculate the area of a circle, which is 3.1416 * radius * radius. If you did this many times in the application, undoubtedly you would write a routine called AREA which would do this multiplication and return the answer, with only the radius as the input to the routine. The normal FORTH practice would be to place the radius on the Data Stack, call AREA , and expect the result to be returned on top of the Data Stack. This sequence of operations might be diagrammed as in Figure 1 (the numbers in parenthesis show the order of execution).

```
 (1)        (2)        (3)        (4)        (5)

radius      AREA       DUP       radius       *
------     radius     radius     radius     radius
           ------     ------     ------     radius
                                            ------

      (6)               (7)             (8)          (9)

radius squared          PI               *           area
--------------    radius squared         PI          ----
                  --------------   radius squared
                                   --------------
```

Figure 1. A simplified Data Stack diagram illustrating the operation of the definition
: AREA (radius -- area)
DUP * PI * ;

The words AREA , DUP , * , and PI represent whatever operations these words do. It would make the example much too complicated to try to include all of their stack operations within Figure 1.

As you can see, calculating the area is rather complicated as far as FORTH is concerned, but really very easy for the programmer. And it is made even easier by transferring parameters on the Data Stack.

In any case, notice in the figure that the Data Stack always grows from the top. Some of these operations consume two or three items on the stack, but they always return only one item back to the top of the stack.

An alternate procedure for accomplishing the same thing is shown in Figure 2, where variables are used for parameter passing.

```
(1)  radius RADIUS !
(2)  : AREA  ( -- )
        RADIUS @
        DUP  *  PI  *
        ( area )  AREA-STORAGE  ! ;
```

Figure 2. An alternate to the method of Figure 1 in which variables are used to transfer parameters.

Really the only difference between these two schemes is that the programmer has to be conscious of the overhead of transferring the parameters in the second method, but that is all done automatically by the machine in the first method.

This is only a very elementary, and possibly obvious, example of using variables instead of the Data Stack for parameter passing. However, it does show the idea Don't be afraid to use variables until you become comfortable with the Data Stack. Your programs will run slower and be longer, but they will execute.

There is great power in the operation of the Data Stack because it controls the order in which things can happen. This can most easily be illustrated by another simplified operation diagram. Consider the two definitions:

```
:AA 2 3 + 5 * 7 - 6 / ;
```

and

```
:BB 2 3 5 7 6 + * - / ;
```

As you can see from Figure 3 and Figure 4, these equations do not produce the same result.

```
(1)    (2)    (3)    (4)    (5)    (6)    (7)

AA      2      3      +      5      5      *
--     ---     2      3     ---     5      5
              ---     2            ---     5
                     ---                  ---

(8)    (9)    (10)   (11)   (12)   (13)   (14)

25      7      -     18      6      /      3
---    25      7     ---    18      6     ---
       ---    25            ---    18
              ---                  ---
```

Figure 3. A simplified Data Stack diagram illustrating the operation of the definition
: AA (-- n)
2 3 + 5 * 7 - 6 / ;

```
(1)    (2)    (3)    (4)    (5)    (6)    (7)

BB      2      3      5      7      6      +
---    ---     2      3      5      7      6
              ---     2      3      5      7
                     ---     2      3      5
                            ---     2      3
                                   ---     2
                                          ---

(8)    (9)    (10)   (11)   (12)   (13)   (14)

13      *     65      -    -62      /     -1
 5     13      3     65      2    -62     ---
 3      5      2      3     ---     2
 2      3     ---     2            ---
---     2            ---
       ---
```

Figure 4. A simplified Data Stack diagram illustrating the operation of the definition (FORTH-83)

It has been my experience that most program bugs will show up as problems with the Data Stack during the early part of your learning cycle, but that will diminish rapidly as you gain experience with FORTH. The situation illustrated in Figure 3 and Figure 4 is one of the most common bugs; it results from trying to be too clever with a definition when there is absolutely nothing to gain from it!

Of course, there is another problem illustrated in Figure 4, having to do with a quirk of "floored math",

You would expect the result of the division of 2 by -62 to be 0, which is what you will get in fig-FORTH and FORTH-79, but not in FORTH-83! FORTH-83 uses "floored math", but not fig-FORTH or FORTH-79. Remember, the only way to gain experience is to try; literally, nothing ventured--nothing gained.

The Return Stack

There is another stack in FORTH which is very important, but is usually not manipulated directly by the programmer; this is the Return Stack. Its purpose is identical to the hardware stack familiar to Assembly language programmers. The return addresses are stored on this stack, and some very fancy and tricky programming can be done by manipulating the Return Stack. However, be careful and be sure that you know what you are doing, because your whole program can blow up in your face with only a simple mistake here!

Usually, the only time a FORTH programmer directly uses the Return Stack is when he needs some very temporary storage. The word, >R , is used to remove the top word of the Data Stack and send it to the top of the Return Stack; the word, R> , does exactly the opposite. Therefore, these two words are used for temporarily storing a number and then recovering it before the end of a definition. >R >R will store a double-word and R> R> will recover it. Remember tha anything put onto the Return Stack must be removed before the end of a definition, or the program will return to the wrong place at the end of the operation.

There is a word, EXIT , which strips the top of the Return Stack so that the program sequence will return to the next previous calling routine, rather than to the one which called the word containing EXIT . EXIT can be crudely defined as R> DROP . Think for a moment about what that does to stuctured programming and shudder. It is very much like a pseudo GOTO .

FORTH makes a lot of transparent use of the Return Stack, so don't be surprised if you run into some hard-to-find bugs, if you blithely go where angels fear to tread.

Other FORTH Stacks

FORTH normally does not use any other stacks, but new ones can be created for special cases. If you are doing a lot of string manipulation you might want to do it on a stack, or you might want to build a separate stack for processing floating-point math.

These "stacks" are usually multi-dimensioned arrays with special words for storing and removing data, so that they look like a stack to the programmer, but a standard array to FORTH. It probably would be possible to shift the U and S pointers so as to create real stacks, but I don't know of anyone who wants to take that chance, when the other method works so well. With the 6809, there is little difference between the time for doing a FORTH stack operation or a FORTH array operation, so there is little incentive to do the extra work of having additional true stacks.

The advantage of extra "stacks" is that you can do any sort of manipulations on them that you might want with a much reduced chance of committing a fatal error. One problem with using the Data Stack is knowing when enough is enough. If you put too much onto the Data Stack, you run the risk of losing your place and not knowing which important number or pointer is where. Therefore, separate stacks for the program housekeeping and for string pointers can be a very good idea.

Another use for a separate stack would be for parameter passing for a Pascal interpreter (not compiler) written in FORTH. I have often thought that I might go back to Pascal or C for some jobs if it were not such a pain to go through the edit-compile-debug cycle. Once the program was satisfactorily debugged, it could then be compiled in the normal way. I may write such a program one of these days, just for the challenge; I don't think that it would be too difficult.

Benchmark Gone Awry?

I want to share with you something which I find amusing and absolutely not to be taken seriously!

I was reading the adds for the MUSTANG-020 and the MUSTANG-08, and was attracted by the benchmark tests for the 1,000,000 iterations of the empty loop in C, so I decided to try the benchmark in FORTH. Much to my surprise, with the definition shown in Figure 5, I got the following results:

FF9 and FLEX at 1 MHz 39 sec
LMI Z-80 FORTH and CP/M at 4 MHz 37 sec

Does this mean that a 6809 running at 4MHz would be just as fast as a 10 MHz 68008? **If that were true, could a 6809 at 20 MHz match a 68020 at 16 MHz? Maybe, Motorola quit too soon on the 6809!

When you couple these results with the virtually 0 sec compile time of FORTH, the possibilities of a faster 6809/FORTH fairly boggle the mind!

Too bad I missed getting this into the April issue.

```
: BB  ( -- n )
   2  3  5  7  6  +  *  -  /  ;

: BENCHMARK    ( -- )                      \ RDL 04/03/87
   16  0  DO                         \ FORTH-83 has no
     62500  0  DO                    \ provision for a loop
     LOOP                            \ counter of 1,000,000
   LOOP ;
```

Figure 5. A FORTH definition to do 1,000,000 iterations of an empty loop.

** Editor's Note: I would guess that it is more a function of efficiency of compiled code. For crunching numbers I believe FORTH is hard to beat. FORTH was originally designed to do precise mathamatical functions for one of the worlds most advanced glass lens telescopes. Math is it's "thing".

I agree, a high speed 6800 would have been a whiz. I have heard of some engineers and marketing types within Motorola who feel that Motorola let the ball drop by not proceeding with the 6809 as a higher speed chip. Actually I have a 6809 that runs much faster, but then that is another tale. In some applications, 8 bits is better than any larger number of bits. But then we all have to have the "bigger" because it's "better", right? Or is it?...

DMW

P68000 uLAB™

"One of the best training tools I have ever seen & very practical also!"

Last month I reviewed an excellent prototype S50 (or stand alone system with external power supply) for you 6809 buffs. This month I want to tell you about a 68000 development system. I have had a ball messing with this thing. And — what is even better I learned a lot about 68000 programming and interfacing. This little beauty is the first computer I have ever taken to bed with me!

When the P68000 u LAB first arrived, I fell for it. It is one of the neatest, compact systems, for learning or experimenting, I have ever seen. As most of you know, I have seen more than several. *This is one item that far exceeds it's advertising claims. Even the picture doesn't do it justice.* **It's a lot of learning and fun for a small outlay.**

But first, a little bit of background. This system is manufactured and sold by Quasitronics, Inc., under license from the University of Pittsburg, University Research and Development Association, Inc. (URDA).

URDA was formed to allow its principals and employees an interface to interact with industry, university and government agencies, and to provide services in a number of technological fields. This system is just one example of that project. It should be something that anyone interested in learning about the 68000 series of microprocessors would be interested in acquiring. Especially considering the low price and high quality of this system.

The one thing I found disquieting is the quality of the printer used in preparing the otherwise excellent documentation. It appears to have been done on a Macintosh and printed out on a dot printer. For some reason, I don't get with some other products, it seems that an item of this level of quality deserves better. Fact is, I like it so much that I will print the manual on the LaserWriter™, just to make things even (if furnished in a Macintosh compatible text file). It is somewhat akin to marrying a beautiful lady only to discover on the honeymoon she wears a wig and is baldheaded — with warts all over her head. Distracting, to say the least.

Now that I have that off my chest, on to more interesting observations.

When you first open the folder you are impressed with the planning in laying out the 68000 system and the loose leaf binder. The binder is three sections of heavy duty vinyl with the computer bolted to the inside third cover. So what you have is a three section unit, two pages of documentation (if beyond the first page) on the left and the computer on the right. All in one compact package. The only separate thing is the wall plug power adapter, and the optional wirewrap system if you ordered that also (which I recommend).

The system is totally self contained. It was developed to enable a student an economical way of learning about the 68000, and more important, having it available at times regular lab systems would be unavailable. Of course this just doesn't apply to students, I know many professionals who would benefit from owning one of these little beauties. I guess by this time you must realize I really like mine. I have talked to several others who also own one and we all share the same level of appreciation for the P68000 uLAB.

Many of the descriptions given here are excerpted directly from the manual. Normally I don't do that, but in this case I find no better way to put it.

The package is really a "Notebook Computer", containing a 68000 CPU, and all necessary support (clock, memory, software) on a circuit board with a 28 key multi-purpose keyboard, 8 digit 7 segment hexadecimal LED display, tape I/O, calculator type power supply, users manual, programmers reference manual, complete schematics, completely commented operating system and utility listings.

The users manual contains a complete description of the design criteria, construction and operation, including examples of all key functions, memory map, software utilities, sound generation, visual display, timing clock examples and enlightening and nontrivial exercises for both the beginner and old pro.

This is one of a series of educational tools to be offered by URDA, and if this is an example, the others should also be as well accepted.

An add-on optional expansion unit is available to enable the user to hardwire the unit to actual physical devices — LED's, switches, motors, actuators, etc. We received the PWWEB-A wire wrap expansion board kit which includes, among other

Telephone: (615) 842-4600 — **South East Media** — **Telex: 5106006630**

*OS-9, UniFLEX, FLEX, SK*DOS*

SOFTWARE

!!! Please Specify Your Operating System and Disk Size !!!

SCULPTOR

Full OEM & Dealer Discounts Available!

THE SCULPTOR SYSTEM
Sculptor combines a powerful fourth generation language with an efficient database management system. Programmers currently using traditional languages such as Basic and Cobol will be amazed at what Sculptor does to their productivity. With Sculptor you'll find that what used to take a week can be achieved in just a few hours.

AN ESTABLISHED LEADER
Sculptor was developed by professionals who needed a software development tool with capabilities that were not available in the software market. It was launched in 1981 and since then, with feedback from an ever-increasing customer base, Sculptor has been refined and enhanced to become one of the most adaptable, fast, and above all reliable systems on the market today.

SYSTEM INDEPENDENCE
Sculptor is available on many different machines and for most operating systems, including MS DOS, Unix/Xenix and VMS. The extensive list of supported hardware ranges from small personal computers, through multi-user micros up to large minis and mainframes. Sculptor is constantly being ported to new systems.

APPLICATION PORTABILITY
Mobility of software between different environments is one of Sculptor's major advantages. You can develop applications on a stand-alone PC and – without any alterations to the programs – run them on a large multi-user system. For software writers this means that their products can reach a wider marketplace than ever before. It is this system portability, together with high-speed development, that makes Sculptor so appealing to value added resellers, hardware manufacturers and software developers of all kinds.

SPEED AND EFFICIENCY
Sculptor uses a fast and proven indexing technique which provides instant retrieval of data from even the largest of files. Sculptor's fourth generation language is compiled to a compact intermediate code which executes with impressive speed.

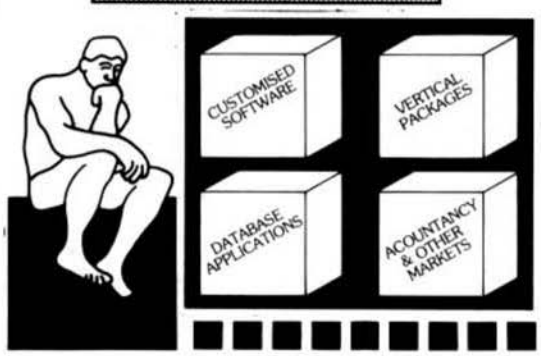
INTERNATIONALLY ACCEPTED
By using a simple configuration utility, Sculptor can present information in the language and format that you require. This makes it an ideal product for software development almost anywhere in the world. Australasia, the Americas and Europe – Sculptor is already at work in over 20 countries.

THE PACKAGE
With every development system you receive:
- ☐ A manual that makes sense
- ☐ A periodic newsletter
- ☐ Screen form language
- ☐ Report generator
- ☐ Menu system
- ☐ Query facility
- ☐ Set of utility programs
- ☐ Sample programs

For resale products, the run-time system is available at a nominal cost.

Facts **Features**

Sculptor for 68020
OS-9 & UniFLEX
$995



DATA DICTIONARY
Each file may have one or more record types described. Fields may have a name, heading, type, size, format and validation list. Field type may be chosen from:
- ☐ alphanumeric
- ☐ integer
- ☐ floating point
- ☐ money
- ☐ date

DATA FILE STRUCTURE
- ☐ Packed, fixed-length records
- ☐ Money stored in lower currency unit
- ☐ Dates stored as integer day numbers

INDEXING TECHNIQUE
Sculptor maintains a B-tree index for each data file. Program logic allows any numbers of alternative indexes to be coded into one other file.

INPUT DATA VALIDATION
Input data may be validated at three levels
- ☐ automatic by field type
- ☐ validation list in data dictionary
- ☐ programmer coded logic

ARITHMETIC OPERATORS
–	Unary minus
*	Multiplication
/	Division
%	Remainder
+	Addition
–	Subtraction

RELATIONAL OPERATORS
=	Equal to
<	Less than
>	Greater than
< =	Less than or equal to
> =	Greater than or equal to
< >	Not equal to
and	Logical and
or	Logical or
ct	Contains
bw	Begins with

SPECIAL FEATURES
- ☐ Full date arithmetic
- ☐ Echo suppression for passwords
- ☐ Terminal and printer independence
- ☐ Parameter passing to sub-programs
- ☐ User definable date format

MAXIMA AND MINIMA
Minimum key length	1 byte
Maximum key length	160 bytes
Minimum record length	3 bytes
Maximum record length	32767 bytes
Maximum fields per record	32767
Maximum records per file	16 million
Maximum files per program	16
Maximum open files	Operating system limit

PROGRAMS
- ☐ Define record layout
- ☐ Create new indexed file
- ☐ Generate standard screen-form program
- ☐ Generate standard report program
- ☐ Compile screen-form program
- ☐ Compile report program
- ☐ Screen-form program interpreter
- ☐ Report program interpreter
- ☐ Menu interpreter
- ☐ Query facility
- ☐ Reformat file
- ☐ Check file integrity
- ☐ Rebuild index
- ☐ Alter language and date format
- ☐ Setup terminal characteristics
- ☐ Setup printer characteristics

SCREEN-FORM LANGUAGE
- ☐ Programmer defined options and logic
- ☐ Multiple files open in one program
- ☐ Default or programmer processing of exception conditions
- ☐ Powerful verbs for input, display and file access
- ☐ Simultaneous display of multiple records
- ☐ Facility to call sub-programs and operating system commands
- ☐ Conditional statements
- ☐ Subroutines
- ☐ Independent of terminal type

MUSTANG-020 Users - Ask For Your Special Discount!

* Tandy CoCo III Special - Reg. $595 * Special $389 *

	*	**	***		*	**	***
MUSTANG-020	$995	$199	$595	PC/XT/AT MSDOS	$595	$119	$595
OS/9 UniFLEX 6809	"	"	"	AT&T 3B1 UNIX	"	"	"
IBM Compatibles	"	"	"	SWTPC 68010 UniF	$1595	$319	$797
Tandy CoCo III	Special $389.00			SWTPC 68010 UNIX	$1990	$398	$995

... Sculptor Will Run On Over 100 Other Types of Machines ...

... Call for Pricing ...

!!! Please Specify Your Make of Computer and Operating System !!!

* Full Developement Package
** Run Time Only
*** C Key File Library

Availability Legends
O = OS-9, S = SK*DOS
F = FLEX, U = UniFLEX
CCO = Color Computer OS-9
CCF = Color Computer FLEX

VISA

South East Media
5900 Cassandra Smith Rd. - Hixson, Tn. 37343
Telephone: (615) 842-4600 Telex: 5106006630

mastercard

** Shipping **
Add 2% U.S.A. (min. $2.50)
Foreign Surface Add 5%
Foreign Airmail Add 10%
Or C.O.D. Shipping Only

*OS-9 is a Trademark of Microware and Motorola-*FLEX and UniFLEX are Trademarks of Technical Systems Consultants-*SK*DOS is a Trademark of Star-K Software Systems Corp.

Telephone: (615) 842-4600 **South East Media** *Telex: 5106006630*

*OS-9, UniFLEX, FLEX, SK*DOS*

FORTHBUILDER is a stand-alone target compiler (crosscompiler) for producing custom Forth systems and application programs. All of the 83-standard defining words and control structures are recognized by FORTHBUILDER.
FORTHBUILDER is designed to behave as much as possible like a resident Forth interpreter/compiler, so that most of the established techniques for writing Forth code can be used without change.
Like compilers for other languages, FORTHBUILDER can operate in "batch mode".
The compiler recognizes and emulates target names defined by CONSTANT or VARIABLE and is readily extended with "compile-time" definitions to emulate specific target words.
FORTHBUILDER is supplied as an executable command file configured for a specific host system and target processor. Object code produced from the accompanying model source code is royalty-free to licensed users.
F, CCF, S - $99.95

DATABASE ACCOUNTING

XDMS from Westchester Applied Business Systems

FOR 6809 FLEX-SK*DOS(5/8")

Up to 32 groups/fields per record! Up to 12 character filed name! Up to 1024 byte records! User defined screen and print control! Process files! Form files! Conditional execution! Process chaining! Upward/ Downward file linking! File joining! Random file virtual paging! Built in utilities! Built in text line editor! Fully session oriented! Enhanced forms! Boldface, Double width, Italics and Underline supported! Written in compact structured assembler! Integrated for FAST execution!

XDMS-IV Data Management System

XDMS-IV is a brand new approach to data management. It not only permits users to describe, enter and retrieve data, but also to process entire files producing customized reports, screen displays and file output. Processing can consist of any of a set of standard high level functions including record and field selection, sorting and aggregation, lookups in other files, special processing of record subsets, custom report formatting, totaling and subtotaling, and presentation of up to three related files as a "database" on user defined output reports.

POWERFUL COMMANDS!

XDMS-IV combines the functionality of many popular DBMS software systems with a new easy to use command set into a single integrated package. We've included many new features and commands including a set of general file utilities. The processing commands are Input-Process-Output (IPO) oriente which allows almost instant implementation of a process design.

SESSION ORIENTED!

XDMS-IV is session oriented. Enter "XDMS" and you are in instant command of all the features. No more waiting for a command to load in from disk! Many commands are immediate, such as CREATE (file definition), UPDATE (file editor), PURGE and DELETE (utilities). Others are process commands which are used to create a user process which is executed with a RUN command. Either may be entered into a "process" file which is executed by an EXECUTE statement. Processes may execute other processes, or themselves, either conditionally or unconditionally. Menus and screen prompts are easily coded, and entire user applications can be run without ever leaving XDMS-IV

IT'S EASY TO USE!

XDMS-IV keeps data management simple! Rather than design a complex DBMS which hides the true nature of the data, we kept XDMS-IV file oriented. The user view of data relationships is presented in reports and screen output, while the actual data resides in easy to maintain files. This aspect permits customized presentation and reports without complex redefinition of the database files and structure. XDMS-IV may be used for a wide range of applications from simple record management systems (addresses, inventory ...) to integrated database systems (order entry, accounting...)
The possibilities are unlimited...

FOR 6809 FLEX-SK*DOS(5/8") $249.95

ASSEMBLERS

ASTRUK09 from S.E. Media -- A "Structured Assembler for the 6809" which requires the TSC Macro Assembler.
F, S, CCF - $99.95

Macro Assembler for TSC -- The FLEX, SK*DOS STANDARD Assembler.
Special -- CCF $35.00; F, S $50.00

OSM Extended 6809 Macro Assembler from Lloyd I/O. -- Provides local labels, Motorola S-records, and Intel Hex records; XREF. GeneÒrate OS-9 Memory modules under FLEX, SK*DOS.
*FLEX, SK*DOS, CCF, OS-9 $99.00*

Relocating Assembler/Linking Loader from TSC. -- Use with many of the C and Pascal Compilers.
F, S, CCF $150.00

MACE, by Graham Trott from Windrush Micro Systems -- Co-Resident Editor and Assembler; fast interactive A.L. Programming for small to medium-sized Programs.
F, S, CCF - $75.00

XMACE -- MACE w/Cross Assembler for 6800/1/2/3/8
F, S, CCF - $98.00

Availability Legends
O = OS-9, S = SK*DOS
F = FLEX, U = UniFLEX
CCO = Color Computer OS-9
CCF = Color Computer FLEX

VISA

South East Media
5900 Cassandra Smith Rd. - Hixson, Tn. 37343

master card

**** Shipping ****
Add 3% U.S.A. (min. $2.50)
Foreign Surface Add 5%
Forel Airmail Add 10%
Or C.O.D. Shipping Only

*OS-9 is a Trademark of Microware and Motorola-*FLEX and UniFLEX are Trademarks of Technical Systems Consultants-*SK*DOS is a Trademark of Star-K Software Systems Corp.

Telephone: (615) 842-4600

South East Media

Telex: 5106006630

OS-9, UniFLEX, FLEX, SK*DOS

SOFTOOLS

The following programs are included in object form for immediate application. PL/9 source code available for customization.

READ-ME Complete instructions for initial set-up and operation. Can even be printed out with the included text processor.
CONFIG one time system configuration.
CHANGE changes words, characters, etc. globally to any text type file.
CLEANTXT converts text files to standard FLEX, SK*DOS files.
COMMON compare two text files and reports differences.
COMPARE another check file that reports mis-matched lines.
CONCAT similar to FLEX, SK*DOS append but can also list files to screen.
DOCUMENT for PL/9 source files. Very useful in examining parameter passing aspects of procedures.
ECHO echos to either screen or file.
FIND an improve find command with "pattern" matching and wildcards. Very useful.
HEX dumps files in both hex and ASCII.
INCLUDE a file copy program that will accept "includes" of other disk files.
KWIC allows rotating each word, on each line to the beginning. Very useful in a sort program, etc.
LISTDIR a directory listing program. Not super, but better than CAT.
MEMSORT a high-speed text file sorter. Up to 10 fields may be sorted. Very fast. Very useful.
MULTICOL width of page, number of columns may be specified. A MUST!
PAGE similar to LIST but allows for a page header, page width and depth. Adjust for CRT screen or printer as set up by CONFIG. A very smart print driver. Allows printer control commands.
REMOVE a fast file deleter. Careful, no prompts issued. Zap, and its gone!
SCREEN a screen listing utility. Word wraps text to fit screen. Screen depth may be altered at run time.
SORT a super version of MEMSORT. Ascending/descending order, up to 10 keys, case over-ride, sort on nth word and sort on characters if file is small enough, sorts in RAM. If large file, sort is constrained to size of your largest disk capacity.
TPROC a small but nice text formatter. This is a complete formatter and has functions not found in other formatters.
TRANSLIT sorts a file by x keyfields. Checks for duplications. Up to 10 key files may be used.
UNROTATE used with KWIC this program reads an input file and unfolds it a line at a time. If the file has been sorted each word will be presented in sequence.
WC a word count utility. Can count words, characters or lines.

NOTE: this set of utilities consists of 6 5-1/4" disks or 2 8" disks, w/ source (PL9). 3 5-1/4" disks or 1 8" disk w/o source.
Complete set SPECIAL INTRO PRICE:
5-1/4" w/source FLEX - SK*DOS - $129.95
w/o source - $79.95
8" w/source - $79.95 - w/o source $49.95

FULL SCREEN FORMS DISPLAY from Computer Systems Consultants -- TSC Extended BASIC program supports any Serial Terminal with Cursor Control or Memory-Mapped Video Displays; substantially extends the capabilities of the Program Designer by providing a table-driven method of describing and using Full Screen Displays.
F, S and CCF, U - $25.00, w/ Source - $50.00

SOLVE from S.E. Media - OS-9 Levels I and II only. A Symbolic Object/Logic Verification & Examine debugger. Including inline debugging, disassemble and assemble. SOLVE IS THE MOST COMPLETE DEBUGGER we have seen for the 6809 OS-9 series! SOLVE does it all! With a rich selection of monitor, assembler, disassembler, environmental, execution and other miscellaneous commands, SOLVE is the MOST POWERFUL tool-kit item you can own! Yet, SOLVE is simple to use! With complete documentation, a snap! Everyone who has ordered this package has raved! See review - 68 Micro Journal - December 1985. No 'blind' debugging here, full screen displays, rich and complete in information presented. Since review in 68 Micro Journal, this is our fastest mover!
Levels I & II only - OS-9 $69.95

DISK UTILITIES

OS-9 VDisk from S.E. Media -- For Level I only. Use the Extended Memory capability of your SWTPC or Gimix CPU card (or similar format DAT) for FAST Program Compiles, CMD execution, high speed inter-process communications (without pipe buffers), etc. - SAVE that System Memory. Virtual Disk size is variable in 4K increments up to 960K. Some Assembly Required.
Level I OS-9 obj. $79.95; w/ Source $149.95

O-F from S.E. Media -- Written in BASIC09 (with Source), includes: REFORMAT, a BASIC09 Program that reformats a chosen amount of an OS-9 disk to FLEX, SK*DOS Format so it can be used normally by FLEX, SK*DOS; and FLEX, a BASIC09 Program that does the actual read or write function to the special O-F Transfer Disk; user-friendly menu driven. Read the FLEX, SK*DOS Directory, Delete FLEX, SK*DOS Files, Copy both directions, etc. FLEX, SK*DOS users use the special disk just like any other FLEX, SK*DOS disk
O - 6809/68000 $79.95

LSORT from S.E. Media - A SORT/MERGE package for OS-9 (Level I & II only). Sorts records with fixed lengths or variable lengths. Allows for either ascending or descending sort. Sorting can be done in either ASCII sequence or alternate collating sequence. Right, left or no justification of data fields available. LSORT includes a full set of comments and errors messages.
OS-9 $85.00

Availability Legends
O = OS-9, S = SK*DOS
F = FLEX, U = UniFLEX
CCO = Color Computer OS-9
CCF = Color Computer FLEX

VISA

South East Media
5900 Cassandra Smith Rd. - Hixson, Tn. 37343

master card

**** Shipping ****
Add 2% U.S.A. (min. $2.50)
Foreign Surface Add 5%
Foreign Airmail Add 10%
Or C.O.D. Shipping Only

HIER from S.E. Media - *HIER is a modern hierarchal storage system for users under FLEX, SK*DOS.* It answers the needs of those who have hard disk capabilities on their systems, or many files on one disk - any size. **Using HIER a regular (any) FLEX, SK*DOS** disk (8 - 5 - hard disk) can have sub directories. By this method the problems of assigning unique names to files is less burdensome. Different files with the exact same name may be on the same disk, as long as they are in different directories. For the winchester user this becomes a must. Sub-directories are the modem day solution that all current large systems use. Each directory looks to FLEX, **SK*DOS like a regular file, except they have the extension** '.DIR'. A full set of directory handling programs are included, making the operation of HIER simple and straightforward. A special install package is included to install HIER to your particular version of FLEX, SK*DOS. Some assembly required. Install indicates each byte or reference change needed. Typically - 6 byte changes in source (furnished) and one assembly of HIER is all that is required. No programming required!

*FLEX - SK*DOS $79.95*

COPYMULT from S.E. Media -- Copy LARGE Disks to several smaller disks. FLEX, SK*DOS utilities allow the backup of ANY size disk to any SMALLER size diskettes (Hard Disk to floppies, 8" to 5", etc.) by simply inserting diskettes as requested by COPYMULT. No fooling with directory deletions, etc. COPYMULT.CMD understands normal "copy" syntax and keeps up with files copied by maintaining directories for both host and receiving disk system. Also includes BACKUP.CMD to download any size "random" type file; RESTORE.CMD to restructure copied "random" files for copying, or recopying back to the host system; and FREELINK.CMD as a "bonus" utility that "relinks" the free chain of floppy or hard disk, eliminating fragmentation.

Completely documented Assembly Language Source files included.

*ALL 4 Programs (FLEX, SK*DOS, 8" or 5") $99.50*

COPYCAT from Lucidata -- *Pascal NOT required.* Allows reading TSC Mini-FLEX, SK*DOS, SSB DOS68, and Digital Research CP/M Disks while operating under SK*DOS, FLEX1.0, FLEX 2.0, or FLEX 9.0 with 6800 or 6809 Systems. COPYCAT will not perform miracles, but, between the program and the manual, you stand a good chance of accomplishing a transfer. Also includes some Utilities to help out. Programs supplied in Modular Source Code (Assembly Language) to help solve unusual problems.

F, S and CCF 5" - $50.00 F, S 8" - $65.00

VIRTUAL TERMINAL from S.E. Media - Allows one terminal to do the work of several. The user may start as many as eight task on one terminal, under *VIRTUAL TERMINAL* and switch back and forth between task at will. No need to exit each one; just jump back and forth. Complete with configuration program. The best way to keep up with those background programs.

O & CCO - obj. only - $49.95

FLEX, SK*DOS DISK UTILITIES from Computer Systems Consultants -- Eight (8) different Assembly Language (w/ Source Code) FLEX, SK*DOS Utilities for every FLEX, SK*DOS Users Toolbox: Copy a File with CRC Errors; Test Disk for errors; Compare two Disks; a fast Disk Backup Program; Edit Disk Sectors; Linearize Free-Chain on the Disk; print Disk Identification; and Sort and Replace the Disk Directory (in sorted order). -- PLUS -- Ten XBASIC Programs including: A BASIC Resequencer with EXTRAs over "RENUM" like check for missing label definitions, processes Disk to Disk instead of in Memory, etc. Other programs Compare, Merge, or Generate Updates between two BASIC Programs, check BASIC Sequence Numbers, compare two unsequenced files, and 5 Programs for establishing a Master Directory of several Disks, and sorting, selecting, updating, and printing paginated listings of these files. A BASIC Cross-Reference Program, written in Assembly Language, which provides an X-Ref Listing of the Variables and Reserved Words in TSC BASIC, XBASIC, and PRECOMPILER BASIC Programs.

ALL Utilities include Source≥ (either BASIC or A.L. Source Code).

F, S and CCF - $50.00

BASIC Utilities ONLY for UniFLEX -- $30.00

COMMUNICATIONS

CMODEM Telecommunications Program from Computer Systems Consultants, Inc. -- Menu-Driven; supports Dumb-Terminal Mode, Upload and Download in non-protocol mode, and the CP/M "Modem7" Christensen protocol mode to enable communication capabilities for almost any requirement. Written in "C".

*FLEX, SK*DOS, CCF, OS-9, UniFLEX, 68000 & 6809h*

Source $100.00 - without Source $50.00

X-TALK from S.E. Media - X-TALK consists of two disks and a special cable, the hookup enables a 6809 SWTPC computer to dump UniFLEX files directly to the UniFLEX MUSTANG-020. This is the ONLY currently available method to transfer SWTPC 6809 UniFLEX files to a 68000 UniFLEX system. Gimix 6809 users may dump a 6809 UniFLEX file to a 6809 UniFLEX five inch disk and it is readable by the MUSTANG-020. The cable is specially prepared with internal connections to match the non-standard SWTPC SO/9 I/O Db25 connectors. A special SWTPC S+ cable set is also available. Users should specify which SWTPC system he/she wishes to communicate with the MUSTANG-020. The X-TALK software is furnished on two disks. One eight inch disk contains S.E. Media modem program C-MODEM (6809) and the other disk is a MUSTANG-020 five inch disk with C-MODEM (68020). Text and binary files may be directly transferred between the two systems. The C-MODEM programs are unaltered and perform as excellent modem programs also. X-TALK can be purchased with or without the special cables, but this special price is available to registered MUSTANG-020 users only.

X-TALK Complete (cable, 2 disks) $99.95

X-TALK Software (2 disks only) $69.95

X-TALK with CMODEM Source $149.95

Telephone: (615) 842-4600 — **South East Media** — Telex: 5106006630

OS-9, UniFLEX, FLEX, SK*DOS

XDATA from S.E. Media - A COMMUNICATION Package for the UniFLEX Operating System. Use with CP/M, Main Frames, other UniFLEX Systems, etc. Verifies Transmission using checksum or CRC; Re-Transmits bad blocks, etc.
U - $299.99

EDITORS & WORD PROCESSING

JUST from S.E. Media — Text Formatter developed by Ron Anderson; for Dot Matrix Printers, provides many unique features. Output "Formatted" Text to the Display. Use the FPRINT.CMD supplied for producing multiple copies of the "Formatted" Text on the Printer INCLUDING IMBEDDED PRINTER COMMANDS (very useful at other times also, and worth the price of the program by itself). "User Configurable" for adapting to other Printers (comes set up for Epson MX-80 with Graftrax); up to ten (10) imbedded "Printer Control Commands". Compensates for a "Double Width" printed line. Includes the normal line width, margin, indent, paragraph, space, vertical skip lines, page length, page numbering, centering, fill, justification, etc. Use with PAT or any other editor.
* Now supplied as a two disk set:
Disk #1: JUST2.CMD object file,
*JUST2.TXT PL9 source:FLEX, SK*DOS - CC*
Disk #2: JUSTSC object and source in C:
*FLEX, SK*DOS - OS9 - CC*
The JTSC and regular JUST C source are two separate programs. JTSC compiles to a version that expects TSC Word Processor type commands, (.pp .sp .ce etc.) Great for your older text files. The C source compiles to a standard syntax JUST.CMD object file. Using JUST syntax (.p .u .y etc.) With all JUST functions plus several additional printer formatting functions. Reference the JUSTSC C source. For those wanting an excellent BUDGET PRICED word processor, with features none of the others have. This is it!
Disk (1) - PL9 FLEX only. F, S & CCF - $49.95
Disk Set (2) - F, S & CCF & OS9 (C version) - $69.95
OS-9 68K000 complete with Source - $79.95

PAT from S.E. Media - A full feature screen oriented TEXT EDITOR with all the best of "PIE™". For those who swore by and loved only PIE, this is for you! All PIE features and much more! Too many features to list. And if you don't like these, change or add your own. PL-9 source furnished. "C" source available soon. Easily configured to your CRT, with special config section.
*Regular FLEX, SK*DOS $129.50*
• SPECIAL INTRODUCTION OFFER • $79.95
SPECIAL PAT/JUST COMBO (w/source)
*FLEX, SK*DOS $99.95*
OS-9 68K Version $229.00
SPECIAL PAT/JUST COMBO 68K $249.00
Note: *JUST in "C" source available for OS-9*

CEDRIC from S.E. Media - A screen oriented TEXT EDITOR with availability of 'MENU' aid. Macro definitions, configurable 'permanent definable MACROS' - all standard features and the fastest 'global' functions in the west. A simple, automatic terminal config program makes this a real 'no hassel' product. Only 6K in size, leaving the average system over 165 sectors for text buffer - appx. 14,000 plus of free memory! Extra fine for programming as well as text.
*FLEX, SK*DOS $69.95*

BAS-EDIT from S.E. Media - A TSC BASIC or XBASIC screen editor. Appended to BASIC or XBASIC, BAS-EDIT is transparent to normal BASIC/XBASIC operation. Allows editing while in BASIC/XBASIC. Supports the following functions: OVERLAY, INSERT and DUP LINE. Make editing BASIC/XBASIC programs SIMPLE! A GREAT time and effort saver. Programmers love it! NO more retyping entires lines, etc. Complete with over 25 different CRT terminal configuration overlays.
*FLEX, CCF, SK*DOS $39.95*

SCREDITOR III from Windrush Micro Systems -- Powerful Screen-Oriented Editor/Word Processor. Almost 50 different commands; over 300 pages of Documentation with Tutorial. Features Multi-Column display and editing, "decimal align" columns (AND add them up automatically), multiple keystroke macros, even/odd page headers and footers, imbedded printer control codes, all justifications, "help" support, store common command series on disk, etc. Use supplied "set-ups", or remap the keyboard to your needs. Except for proportional printing, this package will DO IT ALL!
*6800 or 6809 FLEX, SK*DOS or SSB DOS, OS-9 - $175.00*

SPELLB "Computer Dictionary" from S.E. Media — OVER 150,000 words! *Look up a word from within your Editor or Word Processor (with the SPH.CMD Utility which operates in the FLEX, SK*DOS UCS).* Or check and update the Text after entry; ADD WORDS to the Dictionary, "Flag" questionable words in the Text, "View a word in context" before changing or ignoring, etc. SPELLB first checks a "Common Word Dictionary", then the normal Dictionary, then a "Personal Word List", and finally, any "Special Word List" you may have specified. SPELLB also allows the use of Small Disk Storage systems.
F, S and CCF - $129.95

STYLO-GRAPH from Great Plains Computer Co. — A full-screen oriented WORD PROCESSOR — (uses the 51 x 24 Display Screens on CoCo FLEX/SK*DOS, or PBJ Wordpak). Full screen display and editing; supports the Daisy Wheel proportional printers.
NEW PRICES 6809 CCF and CCO - $99.95,
F, S or O - $179.95, U - $299.95

Availability Legends
O = OS-9, S = SK*DOS
F = FLEX, U = UniFLEX
CCO = Color Computer OS-9
CCF = Color Computer FLEX

VISA

South East Media
5900 Cassandra Smith Rd. - Hixson, Tn. 37343

master card

** Shipping **
Add 2% U.S.A. (min. $2.50)
Foreign Surface Add 5%
Foreign Airmail Add 10%
Or C.O.D. Shipping Only

Telephone: (615) 842-4600 **South East Media** Telex: 5106006630

OS-9, UniFLEX, FLEX, SK*DOS

STYLO-SPELL from Great Plains Computer Co. -- Fast Computer Dictionary. Complements Stylograph.
NEW PRICES 6809 CCF and CCO - $69.95,
F, S or O - $99.95, U - $149.95

STYLO-MERGE from Great Plains Computer Co. -- Merge Mailing List to "Form" Letters, Print multiple Files, etc., through Stylo.
NEW PRICES 6809 CCF and CCO - $59.95,
F, S or O - $79.95, U - $129.95

STYLO-PAK --- Graph + Spell + Merge Package Deal!!!
F, S or O - $329.95, U - $549.95
O, 68000 $695.00

MISCELLANEOUS

TABULA RASA SPREADSHEET from Computer Systems Consultants -- TABULA RASA is similar to DESKTOP/PLAN; provides use of tabular computation schemes used for analysis of business, sales, and economic conditions. Menu-driven; extensive report-generation capabilities. Requires TSC's Extended BASIC.
F, S and CCF, U - $50.00, w/ Source - $100.00

DYNACALC -- Electronic Spread Sheet for the 6809 and 68000.
F, S, OS-9 and SPECIAL CCF - $200.00, U - $395.00
OS-9 68K - $595.00

FULL SCREEN INVENTORY/MRP from Computer Systems Consultants -- Use the Full Screen Inventory System/Materials Requirement Planning for maintaining inventories. Keeps item field file in alphabetical order for easier inquiry. Locate and/or print records matching partial or complete item, description, vendor, or attributes; find backorder or below stock levels. Print-outs in item or vendor order. MRP capability for the maintenance and analysis of Hierarchical assemblies of items in the inventory file. Requires TSC's Extended BASIC.
F, S and CCF, U - $50.00, w/ Source - $100.00

FULL SCREEN MAILING LIST from Computer Systems Consultants -- The Full Screen Mailing List System provides a means of maintaining simple mailing lists. Locate all records matching on partial or complete name, city, state, zip, or attributes for Listings or Labels, etc. Requires TSC's Extended BASIC.
F, S and CCF, U - $50.00, w/ Source - $100.00

DIET-TRAC Forecaster from S.E. Media -- An XBASIC program that plans a diet in terms of either calories and percentage of carbohydrates, proteins and fats (C P G%) or grams of Carbohydrate, Protein and Fat food exchanges of each of the six basic food groups (vegetable, bread, meat, skim milk, fruit and fat) for a specific individual. Sex, Age, Height, Present Weight, Frame Size, Activity Level and Basal Metabolic Rate for normal individual are taken into account. Ideal weight and sustaining calories for any weight of the above individual are calculated. Provides number of days and daily calendar after weight goal and calorie plan is determined.
F, S - $59.95, U - $89.95

CROSS ASSEMBLERS

TRUE CROSS ASSEMBLERS from Computer Systems Consultants -- Supports 1802/5, Z-80, 6800/1/2/3/8/11/HC11, 6804, 6805/HC05/146805, 6809/00/01, 6502 family, 8080/5, 8020/1/2/35/C35/39/40/48/C48/49/C49/50/8748/49, 8031/51/8751, and 68000 Systems. Assembler and Listing formats same as target CPU's format. Produces machine independent Motorola S-Text.
*68000 or 6809, FLEX, SK*DOS, CCF, OS-9, UniFLEX*
any object or source each - $50.00
any 3 object or source each - $100.00
Set of ALL object $200.00 - w/source $500.00

XASM Cross Assemblers for FLEX, SK*DOS from S.E. MEDIA -- This set of 6800/1/2/3/5/8, 6301, 6502, 8080/5, and Z80 Cross Assemblers uses the familiar TSC Macro Assembler Command Line and Source Code format, Assembler options, etc., in providing code for the target CPU's.
*Complete set, FLEX, SK*DOS only - $150.00*

CRASMB from LLOYD I/O -- Supports Motorola's, Intel's, Zilog's, and other's CPU syntax for these 8-Bit microprocessors: 6800, 6801, 6303, 6804, 6805, 6809, 6811 (all varieties); 6502, 1802/5, 8048 family, 8051 family, 8080/85, Z8, Z80, and TMS-7000 family. Has MACROS, Local Labels, Label X-REF, Label Length to 30 Chars. Object code formats: Motorola S-Records (text), Intel HEX-Records (text), OS9 (binary), and FLEX, SK*DOS (binary). Written in Assembler ... e.g. Very Fast.

CPU TYPE - Price each:

For:	MOTOROLA	INTEL	OTHER	COMPLETE SET
FLEX9	$150	$150	$150	$399
SK*DOS	$150	$150	$150	$399
OS9/6809	$150	$150	$150	$399
OS9/68K	------	------	------	$432

CRASMB 16.32 from LLOYD I/O -- Supports Motorola's 68000, and has same features as the 8 bit version. OS9/68K Object code Format allows this cross assembler to be used in developing your programs for OS9/68K on your OS9/6809 computer.
*FLEX, SK*DOS, CCF, OS-9/6809 $249.00*

GAMES

RAPIER - 6809 Chess Program from S.E. Media -- Requires FLEX, SK*DOS and Displays on Any Type Terminal. Features: Four levels of play. Swap side. Point scoring system. Two display boards. Change skill level. Solve Checkmate problems in 1-2-3-4 moves. Make move and swap sides. Play white or black. This is one of the strongest CHESS programs running on any microcomputer, *estimated USCF Rating 1600+ (better than most 'club' players at higher levels)*
F, S and CCF - $79.95

Availability Legends
O = OS-9, S = SK*DOS
F = FLEX, U = UniFLEX
CCO = Color Computer OS-9
CCF = Color Computer FLEX

VISA

South East Media
5900 Cassandra Smith Rd. - Hixson, Tn. 37343

master card

** Shipping **
Add 2% U.S.A. (min. $2.50)
Foreign Surface Add 5%
Foreign Airmail Add 10%
Or C.O.D. Shipping Only

items, additional IC's, wire wrap sockets (gold plated pins), light sensor, assorted linear devices and components and assorted connectors, assembled cables, a battery holder for backup, and several other items to keep one busy for quite a few evenings.

The following is a listing of the system and it's various components, as delivered:

1. 68000 CPU - 4 Mhz (remember this is for learning)
2. 8K bytes of EPROM with:
   a. 4K for monitor and utilities
   b. 4K for user expansion
3. 4K of static RAM
4. 5X8 matrix keypad (calculator type) with 12 additional keys user programmable and shifted positions
5. 8 section, 7 segment LED display
6. 4 additional push type keys (button switches):
   a. Hardware Reset
   b. Monitor software
   c. User single bit input
   d. User interrupt
7. 3.57 system clock
8. User 8 bit PIA I/O
9. Two 50 pin dip connectors for access to all 68000 pins, I/O signals, etc.
10. On board 5v, 1A (TTL) power capacity, leaving 1/4 amp for user expansion
11. PCB 7 1/2 inches X 10 inches, double sided, solder masked with silk screened component identification keyed to the system schematic
12. System schematic showing all components and connections
13. Memory space for 6800 type peripherals
14. 2 PIA's for I/O
15. 17 IC's
16. Sockets for all critical components (great feature!) if you ever stuck a wire to the wrong place
17. 9 v DC calculator type wall power supply
18. Software monitor with the following functions:
    a. Read/Scan the keypad
    b. Display data on the LED display
    c. Tone generator
    d. Tape read/write routines
    e. Interrupt/Trap vectoring
    f. Break points and single stepping
19. Hardware interface to inexpensive tape units through two 1/8 inch phono jacks
20. 60 page users manual
21. Motorola Programmers Reference Manual

Optional accessories either available or in development:

a. Expansion wire wrap board
b. Expansion ribbon cables
c. Piggy-back RAM expansion card for up to 16K bytes
d. Communications module to connect multiple P6800 systems

As I do this review, I am pressed, by a friend and former associate, Bob Nay, again a college prof, calling and waiting to borrow and look over my system in view of ordering them for his school*. For him I will loan mine out, **this one time**, but for the rest of you guys, go get your own. I hate to be without mine. I find it invaluable for trying something or another, when I get a "spur of the moment" flash of genius. I haul it around in my van every where I go. Even my dog Charlie is getting jealous of the thing. My wife, bless her heart, has gotten used to my affection (affliction) with nifty computer things, years ago (I think?).

This is one product I heartily recommend. It not only looks and feels great, but it actually works. The LED's light up, the clock ticks (technically it pulses), the keypad responds, the speaker beeps, the CPU understands and responds to 68000 instructions, the tape routines read and write, the RAM does it's thing, what else can I say? *It works!*

I didn't find one significant glitch. And it sure makes one learn the conservative ways of programming, but then we should always do that anyway, right? Also not to be forgotten is the price. It is, simply put, **a bargain & all for less than 200 bucks!**

For additional information or to order your own contact:

Quasitronics
211 Vandale Drive
Houston, PA 15342

1 800 245-4192

* O.K. Bob, it's on the way. From an educators viewpoint let us all know what you think. I'll publish your report when I get my P68000 uLAB back. Have Fun! DMW

A staff Review

EOF

THE ED TEXT EDITOR

A commercially available text editor and formatter system which is written in C and runs on several 68000 systems.

E. M. (Bud) Pass, Ph.D.
Computer Systems Consultants, Inc.
1454 Latta Lane, N. W.
Conyers, GA 30207
404-483-4570/1717

The ED text editor is a program marketed by Meta Media, Inc. (P O Box 292, Atlanta, GA 30301) for OS-9/68K and for PC clones.

It is capable of editing files with arbitrary contents and of arbitrary size (63000 bytes per page). It will support up to nine simultaneous edit sessions of the same or different files.

ED assumes the use of a keyboard similar to that used on PC clones, although this default may be modified during the ED installation process. Following are the default control keys:

```
Q
W        begin line
E        end line
R        screen left
T        screen right
Y        help
U        undo
I
O        previous word
P        next word

A        begin next line
S
D        delete word
F        find and replace
G        goto next find and replace
H
J
K
L

Z        delete to line begin
X        delete line
C        delete to line end
V        insert control character
B        center text
N        repeat next request # times
M

F1       cut and paste
F2       mark and goto mark
F3       status
F4       goto line
F5       scroll screen up
F6       scroll screen down
F7       buffer large file
F8       adjust screen column
F9       move to different window
F10      configuration
```

```
up arrow     move cursor up one line
down arrow   move cursor down one line
left arrow   move cursor left one column
right arrow  move curwor right one column
pgup         display previous screen
pgdn         display next screen
home         top of screen, top of file
end          bottom of screen, bottom of
  file
ins          toggle insert mode
del          delete character under cursor
backspace    delete character to left of
  cursor
tab          move cursor to next tab stop
esc          abort current command
```

As might be expected, the installation process may become quite complex, depending upon the details of the hardware-software combination upon which ED and MF are being installed. This process will be covered later, as it is one of the current problems with the use of the products.

ED provides an extensive online help facility in two ways. The user may enter HELP directly from an operating system command line or may hit CTRL-Y while in the ED editor. In either case, ED displays the first page of the list of topics for which assistance is available. The user selects which of the listed topics is to be expanded by entering enough characters to uniquely identify the topic and hitting RETURN. If this topic has another level of topics, this process may be continued recursively until the desired information is found. At any point, the user may hit the PGUP key to return to the main help topic menu or ESC to exit one menu level.

Unlike many other editors, such as Micro-EMACS, ED does not define the CTRL-S and CTRL-Q keys, which are used by many operating systems, including OS-9/68K and MS-DOS, for terminal flow control, and by many terminals for the same purpose. This allows ED to be used with a wider range of terminals and over communications lines or between computer systems.

THE MF FORMATTER

The MF formatter is a program marketed by Meta Media, Inc. for OS-9/68K and for PC clones. It is meant to accompany the ED text editor, although it could be used independently.

MF has the following capabilities:

```
merges mail lists
processes multiple files
allows up to 16 fonts per line
provides proportional right justification
optionally builds a table of contents and
  index
includes other files into a document
expands user-defined macros in the text
creates text structures such as bullets,
  steps, etc.
outputs multiple part and line headers and
  footers
supports multiple classes of printers
```

MF expands the following sequences when found in the text:

```
|a normal font
|b bold font
|c fixed font
|d italic font
|e equation font
|f footnote font
|gx get information (x=d,f,n,p,t,w)
|p periods and wide spaces to next tab stop
|q periods and thin spaces to next tab stop
|t spaces to next tab stop
|r next word at end at next tab stop
|u toggle underline
|0-|9 user font
|^ superscript
```

MF expands the following commands when found at the left margin in the text:

```
.1-.9          1-9 level heading
.bar nn        output horizontal bar
.body nn       define additional indent
.bullet        define bullet structure
.center xxx    center text
.change xxx    change MF configuration
.define xxx    define macro sequence
.do n          call macro sequence
.exit          terminate structure
.facing nn     shift odd/even pages
.font x        change font
.foot xxx      provide page footing
.format        enable text justification
.hang nnn      define indent for .item
.head xxx      provide page heading
.horizontal nn nn nn control text placement
.image         reset all justification
.indent nn     indent text from left margin
.index xxx     index text
.inline xxx    control inline commands
.item xxx      produce itemized text
.justify       enable flush right
  justification
.left          left-justify centered text
.lpi n         specify lines per inch
.macro x xx    define single-character macro
.merge xxx     specify text merging operation
.need nn       reserve text lines
.nextpage nn   start next page
.note nn       define note structure
.oddpage       start odd-numbered page
.outline nn    define outline structure
.outside       place text on outside of page
.page          start new page
.paragraph xxx define additional indent
.ragged        enable right margin
.right         right-justify centered text
.roman xxx     generate roman/arabic page
  numbers
.run xxx       execute command line
.send xxx      send control sequence to
  printer
.skip nn       skip lines
.source xxx    include file in text
.step x        define step structure
.tab nn .. nn  define tab stops
.toc xxx       add entry to table of contents
.undent nn     indent text from right margin
.underline xx  underline text
.vertical nn nn nn control text placement
.wait          pause at end of page
```

USE OF ED AND MF

The first problem encountered when attempting to use ED or MF is a complex installation procedure. Although it is detailed well in the manual, a non-technical user might find it quite forbidding, especially in cases requiring the setup of terminals or printers not in the list of devices supported by Meta-Media. Meta-Media could materially assist the user with installation scripts, such as are provided by many competing products.

The other problem with ED is that it makes heavy use of the PC clone keyboard function and special keys. Few serial terminals have as many function and special keys as the PC clone keyboard. The user must define alternate key sequences, materially complicating the use of the editor. Other than this problem, ED seems to have few of the arbitrary features which complicate the use of many other text editors and word processors.

Despite these problems, the PC clone or OS-9/68k user with serious word and text processing and formatting requirements should investigate ED and MF.

Bit-Bucket

By: All of us

"Contribute Nothing - Expect Nothing", DMW '86

Continued From Last Month

XBASIC Xplained
or
Things you won't find in the documentation

A MORE POWERFUL COMPILE COMMAND

To those readers who have never tried it, I would recommend going one step further, and instead of COMPILing directly from XBASIC, making use of TSC's exceptionally powerful Pre-Compiler - the XBASIC version being called XPC, for short. It really is a beautiful language in which to write programs, and has several immediate advantages :

1. No line-numbers needed.

2. Meaningful names can be given to Variables or Subroutines, such as :

```
IF WEEK_DAY = THURSDAY THEN GOSUB CALCULATE_PAY
```

3. By using the '\' symbol, long statement-lines can be broken up, and formatted into blocks, eg

```
IF SCALE_WEIGHT = DESIRED_WEIGHT \
THEN GOSUB STOP_INGREDIENT_FLOW \
ELSE GOSUB XMIT_UPDATED_WEIGHT: GOTO WEIGH_MORE
GOSUB XMIT_FINAL_WEIGHT: GOTO NEXT_INGRED
```

Compare this with :

```
10 IF SW=DW THEN GOSUB 1763 ELSE GOSUB 1984: GOTO 1492
20 GOSUB 1066: GOTO 1000
```

Which do you think will have more meaning a year down the road??

4. A useful Library of subroutines can be built up, and included in subsequent programs by library calls, such as $ LIB DEFS.LIB

5. The finally compiled code is exactly the same as that which would have been obtained from the equivalent COMPILEd XBASIC program, but how much easier it is to read the source in the XPC version!

The XPC program itself is written via an EDITOR or word-processor, just as one does with an Assembly-language program, or simply converted from an already existing XBASIC program with the EDITOR. Once the program is written and saved to disk, it is compiled with the command-line below :

+++XPC file-name   or   +++XPC file-name1 file-name2

In the first case, the name of the source-file is also given to the compiled .BAC file, but in the second the compiled file would have the name filename2.BAC. Try it sometime! You won't regret it!!

AN ERROR IN XBASIC

Here is an error in XBASIC which will probably never get fixed by TSC now that they seem to have abandoned the 6809 in favour of the newer 68xxx chips. Maybe you've already experienced it, and been left a little puzzled. What happens is that XBASIC will report ERROR #55 (Unbalanced Parentheses) in Line XXXX, yet a LIST of that Line will show that such is not the case. In actual fact it should have reported ERROR #78 (Undimensioned Array Reference), and usually occurs when a DIM statement has been omitted, followed by an array dimension being out-of-range in a doubly-dimensioned array. To demonstrate this, assume the following :

```
50 DIM A(6,2)
60 PRINT A(7,1)
```

RUNning this program will result in the correct Error-Message ERROR #77 (Array Reference out of range), but if Line 50 is deleted and the program RUN again it will give the incorrect ERROR #55 (Unbalanced Parentheses) AT LINE 60. Probably what happens is that XBASIC abandons its computation of A(7,1) when it finds the first digit out of range, and doesn't proceed beyond the ','. Thus it doesn't see the second paren. If you simply asked it to PRINT A(7) - with no corresponding DIM statement - it will give the correct Error-Message. One of these days, I intend to clean this bug out of the system!!

THE CHAIN STATEMENT

CHAIN is normally only used if we have a program which is much too large to fit in available memory, and yet is structured in such a way that when one section has been executed it is never used again. The program simply moves into its second half and proceeds from there. In such a case I would SPLIT my program into two sections, <game.BAS>, let's say, and <game.NXT>. Note that I like to keep the name of the game the same for both sections, and change the name of the extension only, so they will be listed together in my alphabetical Master-Directory of all files. Herein lies a problem!

Earlier versions of XBASIC, when executing the CHAIN instruction, made the assumption that if a CHAINed file didn't have a .BAC extension then it had to be a regular BASIC type of file and was loaded as such. Somewhere along the way, however, newer versions reversed this assumption (i.e. if the extension wasn't .BAS then it had to be .BAC) and XBASIC would attempt to load it as such, with disastrous results if the program were a .BAS program. Under the latter conditions, the first half of the program would execute OK, but not the second - - unless I gave it the extension .BAS as well, which, of course, meant giving it a different 'game' name.

What to do? Obviously, it meant jumping in and modifying XBASIC once more. My mod will only work for later versions which have this problem, but here it is for what it's worth. It's quite a small change. Somewhere around address $2FAB (6809) locate the code :

```
42 41 26 04 A6 0E 81 53 10 26 D9 5F
B  A                 S                          and change it to :

42 41 26 3038 A6 0E 81 3433 10 3237 D9 5F
B  A                 C
```

SOME ADVANCED PROGRAMMING TECHNIQUES

Let's begin this section by taking a look at a simple Decimal/HEX conversion routine, which you may find useful in its own right :

```
10  H$="": INPUT D
20 REM CONVERT DECIMAL INTEGER TO HEX STRING
30  E=INT(D/16): F=D-E*16: IF F>9 THEN F=F+7
40  H$=CHR%(F+48)+H$: IF E<>0 THEN D=E: GOTO 30
50  PRINT H$: GOTO 10
```

We'll just talk our way around this little program for a while, and see where it leads us. Line 10 initialises the HEX-string, H$, to a NUL, then requests input of a Decimal-Number - any length within reason. Lines 30 and 40 comprise the actual conversion routine, with Line 30 first dividing the decimal-number by 16. The quotient is retained in 'E' and the remainder in 'F' (as a HEX remainder 0 - 15). Finally, Line 30 prepares the remainders 10 - 15 for conversion to the HEX letters A - F by adding 7 to their value. The function of Line 40 is to convert the remainder to a CHR$ and preface this to the previously computed value of H$, returning for more calculation if E has not been eliminated by the current operation. That is, there is at least one more byte to convert. Finally, Line 50 displays the result, and returns to Line 10 for INPUT of another number.

It's not a particularly remarkable program, though it does its job quite well, but it will serve our purpose for what is to come. Before we continue, however, note the useful programming technique of indenting REMs by one space only, and the rest of the program by 2 spaces. This makes it easy to locate all the REMs in your program listing.

There doesn't appear to be much we can do to improve this short 2-line program, does there? Line 40 seems to be forced on us because it MUST be executed whether or not the IF condition in Line 30 modifies F to F+7, so there would appear to be no way we can comfortably tack Line 40 onto the end of Line 30 and still get the program to work properly. Normally, in the IF-THEN situation existing in our example, this would be true, BUT ... there is a neat way to eliminate the IF-THEN altogether, yet still achieve the same effect. We do this by compressing the IF-THEN into a compact logical function *and including it in the CHRS$ function*. Thus :

```
30  E=INT(D/16): F=D-E*16: H$=CHR$(F+48-7*3(F>9)): IF E<>0 THEN D=E: GOTO 30
40  deleted
```

Let's examine the extra enclosure -7*3(F>9) in a little more detail. The part (F>9) makes use of XBASIC's implementation of Boolean logic, whereby the result is -1 if the statement is TRUE, and 0 if it's not. So (F>9) will be replaced by -1 or 0, depending on whether it's TRUE or FALSE, thus making -7*(F>9) equal to either -7*-1, or -7*0, as the case may be. That is, -7*(F>9) becomes +7 if (F>9) is TRUE and 0 if it's FALSE. The end result is that our original CHR$(F+48) will evaluate as CHR$(F+48+7) i.e CHR$(F+55) if (F>9) is TRUE, and to CHR$(F+48-0) if it's FALSE.

If input of 'D' is restricted so as not to exceed 32767, we can both shorten and speed up the program considerably by replacing 'D' with 'D%', 'E' with 'E%', and so on. Further, this would allow E=INT(D/16) to be reduced to E%=D%/16. We'll have more to say about this later, but for now we'll stick with our current subject.

Here's another situation where we can make use of logic functions. Suppose we had the trivial situation where a message had to be TABbed to say, 30, if the response to a question were "Y" (for YES) or to 40 if the response were "N". For example :

```
10 INPUT "Do you like this (Y or N) ",Q$
20 IF Q$ = "Y" THEN PRINT TAB(30); ELSE PRINT TAB(40);
30 PRINT "Good!"
```

In this case, Lines 20 and 30 can be combined so :

```
20 PRINT TAB(30 - 10 * (Q$="N")); "Good!"
```

Note that this will give a minimum TAB of 30, but bump it by 10 if the response is "N" (ie -10*-1 = +10). A "Y" response would evaluate as FALSE, giving -10*0, or 0 change to the TAB of 30. And what a saving in program-length!!

To close this discussion of logic functions, let's imagine the ridiculous situation where say X has to be divided by 10 if Y<=6 otherwise divided by 20. In addition, X has to be multiplied by 11 if Z<=9, otherwise multiplied by 15. Compare this :

```
10 IF Y <= 6 THEN X = X / 10 ELSE X = X / 20
20 IF Z <= 9 THEN X = X / 11 ELSE X = X / 15
```

with this :

```
10 X = X / (10 - 10 * (Y> 6)) * (11 - 4 * (Z> 9))
```

With a little practice, you'll soon find it quite natural to read the '*' as 'IF', and the whole of the new Line 10 as :

Divide X by 10, or by 20 (ie 10+10), IF Y>6 and multiply the result by 11, or by 15 (ie 11+4), IF Z>9.

These are but a few typical examples of an unusual use of logic functions, which should serve, not only as a guide to more complex functions, but perhaps help clear up the interpretation of any such cases you may already have come across.

INTEGERS AND THE INT(X) FUNCTION

You may wonder what I could possibly say about these perfectly understandable aspects of XBASIC that you don't already know, but we'll have a go at it anyway.

In order to get on the same communication wave-length, let's begin by agreeing on a few definitions. Firstly, INT(X), where X is a **non-integer** Floating-Point number **is the integer immediately BELOW that number**. That is to say, INT(5.7) = 5 and INT(-5.7) = -6. It is unfortunate that, as of the date of writing, the popular K-BASIC compiler (which gives one the capability of compiling XBASIC programs into machine-code), changes this definition for negative numbers, thus making their INT(X) into a TRUNCATE(X). As you read on, you'll see that life with INT(X) is complicated enough, without having its meaning re-defined! I wouldn't dream of re-working my complete library of XBASIC programs just to accommodate this non-standard definition.

Secondly, to continue with our discussion, the numbers 1, 13, -15, etc are Integer CONSTANTS, while A%, B% and C% are Integer VARIABLES in the range +/- 32767. The numbers 123.456, 0.987 and PI are FP CONSTANTS, while A, B and C are FP VARIABLES.

OK, now we can get down to some serious business. You'll find that lots of programs can be speeded up (quite apart from taking up less room in memory) if as many variables as possible are made into Integers. As a case in point, let's consider the simple loop :

```
10 FOR I = 1 TO 10
20   PRINT I;
30 NEXT I
```

RUNning this program will produce the following :

```
1 2 3 4 5 6 7 8 9 10
```

and if all the 'I's are changed to 'I%' the result will be exactly the same, only it will happen faster. Now let's add another line to the original program and RUN it again.

```
25   IF I/3 = INT(I/3) THEN PRINT "BOING!";
```

This time we'll see :

```
1 2 3 BOING! 4 5 6 BOING! 7 8 9 BOING! 10
```

but not if we change all 'I's to 'I%' as we did in the first example. This time we'll see "BOING!" printed out after each and every number. What went wrong??

If we tabulate I, I/3 and INT(I/3) we get :

3I	1	2	3	4	5	6	7	8	9	10
I/3	0.33	0.66	31	1.33	1.66	32	2.33	2.66	33	3.33
INT(I/3)	0	0	31	1	1	32	2	2	33	3

causing a match at every 'I' which is exactly divisible by 3, whereas 'I%/3' and 'INT(I%/3)' are both identical to the line for 'INT(I/3)' above. How can we preserve the intent of the original program, and STILL change our 'I's to 'I%'? Unlike I/3, where I is FP and 3 an Integer, producing a FP result, I%/3 produces the sequence 0, 0, 1, 1, 1, 2 etc because I%3 and 3 are both true integers, producing an Integer result. Therefore the solution is to make one of them a FP Integer, which will in turn produce a FP result. We obviously can't do this to I%, but we can change the '3' to '3.' (which XBASIC interprets as a FP '3.0000'). This makes Line 25 :

```
25   IF I%/3. = INT(I%/3) THEN PRINT "BOING!";
```

and we've achieved the desired result. Of course, an astute reader would also observe that we no longer need the INT() part either, so we end up with :

```
25   IF I%/3. = I%/3 THEN PRINT "BOING!";
```

The left-hand side of the equation now produces a FP sequence equivalent to that of I/3 in the table above, while the right-hand side produces the Integer sequence of INT(I/3) above. And so once more we have a match only at every third number.

Another candidate for conversion is the pattern :

```
X = INT(RND(0) * 52 +1)
```

which creates a random integer in the range 1 - 52, as part of a card-shuffling routine, let's say. We would change this to read :

```
X% = RND(0) * 52 +1
```

Note that we conserve memory in two ways - (i) by requiring only 2 bytes to store the value of an Integer as compared to 8 bytes for a FP variable and (ii) by making our program-lines slightly shorter. As mentioned earlier, the main effect will be that of speeding-up program execution, to the point that if there are any delay-loops in your program you may have to adjust the 'target-count' of the loop to bring it back to the original delay-time. Keep an eye open for this possibility!

But ... there are many pitfalls awaiting the unwary - - especially where division is involved, as in our first example above, or if we venture into the domain of negative numbers. Also, what should we do if, instead of I%/3 (which was easy to change to I%/3.) we have the situation I%/J%, where it is **impossible** to tack a decimal-point onto either Integer?

Let's examine these different possibilities by considering one line of a program :

```
100 A = INT(B / 4 * C)
```

where examination of the entire program shows that A, B and C are **always** integers in the range +/-32767. A prime candidate for conversion to A%, B% and C%, and elimination of INT(), you'd think. Let's test this theory by assigning, let's say, the values B = 7 and C = 5. Under these circumstances Line 100 would produce the result A = 8, because INT (7 / 4 * 5) = INT(8.75) = 8. On the other hand, A% = B% / 4 * C% would give the result as 5. (Why?) Changing the '4' to '4.', as we did earlier, will produce the correct result of 8.

All well and good, but what if we were to replace the constant '4' by a new Variable 'D' to which we'll assign the value '4', eg :

```
100 D = 4: A = INT(B / D * C)
```

It makes no difference when compared to the original Line 100, but in our transformed Line we would have :

```
100 D% = 4: A% = B% / D% * C%
```

which would again give the undesired result of '5'. Now what do we do? We have absolutely nowhere to tack on a decimal-point!

A quick and dirty solution would be to change the order of the Variables on the right-hand side of the equation to give :

```
100 D% = 4: A% = B% * C% / D%          (try it, and compare)
```

where division is now the last operation to be carried out. This, however, will only work correctly if the partial-product, B% * C%, does not exceed the range +/-32767, and provided no negative results are produced along the way. Try setting B equal to -7 in the original Line 100, and compare the end result with B% equal to -7 in the transformed Line. Talk about complications!!

A more elegant solution would be to preserve the original order of the variables, but to multiply the **FIRST** variable on the right-hand side by the **identify for multipcation and division.** This is equal to '1' (or '1.' for our purposes), as multiplying, or dividing, a number by '1' leaves it unchanged - that is, it preserves its identity. Similarly the identity for addition and subtraction equals '0', as adding this number to, or subtracting it from, a number leaves the number unchanged. So now we'll have :

```
100 D% = 4: A% = B% * 1. / D% * C%
```

which should entirely satisfy the transformation. On the other hand, as addition is faster than multiplication, maybe we should consider :

A% = (B% + 0.) / D% * C%

Well now, who would have thought we could have had so much to say about a simple thing like INTEGERs? I've possibly overlooked a few other problems somewhere (let me know if you come across any) but this little discussion should at least pinpoint some of them and perhaps offer some guidelines for overcoming them. Anyway, my advice to you is to tread warily when you venture into the field of Integer-Variables and the INT() function, as it could very well tum out to be a mini-field. Don't make any assumptions as to what the result will be, but test your assignments and expressions for a range of conditions before making your changes permanent.

ASC( ), VAL( ), CHR$( ) & STR$( )

These four String-Functions seem to puzzle quite a few people, creating problems in deciding which to use and where it should be used. The definitions in the XBASIC article usually become perfectly clear only when you fully understand these little monsters. So let's begin by considering them as two sets of twins, ASC() and CHR$() comprising one set and VAL() and STR$() the other.

ASC(I$) will return the decimal ASCII value (not HEX) of the first character in I$, so if 1$ = "Hello" then ASC(I$) = 72, the ASCII value of 'H'. But, if coupled with the use of MID$ or RIGHT$, it can be used to give the decimal ASCII equivalent of any single character in the string. Thus ASC(RIGHT$(1$,1)) = 111, which is the ASCII value of 'o', the right-most character of the string.CHR$(I%) is the exact opposite, in that it generates the ASCII character equivalent of I% so that CHR$(72), if output to the screen would display 'H', while CHR$(111) would display 'o'.Keep in mind, however, that this set of twins can handle only one solitary character at a time.

VAL & STR$( ), on the other hand, can cope with multi-character strings. But, and a big 'but', they deal only with numeric strings. So, given I$ = "Hello" and J$ = "123.4", we would find that VAL(I$) = 0 because "Hello" is not a decimal number, but VAL(J$) = 123.4. Similarly, given I = 123.4, then STR$(I) = "123.4". Big deal, you say. So what purpose do they serve if they can handle only decimal numbers, and apparently leave them unchanged at that?

Perhaps the best way to look at these two is to imagine that VAL is a machine capable of transforming a picture of something into the real thing, while STR$ is an instant camera capable of producing a picture of a real object. Thus, if we had a short-tailed, black-spotted dog from which we wished to produce a long-tailed, zebra-striped pup (and not having very great medical or surgical skills!) we would first of all take a picture of the original. Then we would touch up the photograph appropriately (we are much better artists than we are plastic-surgeons), feed it through our VAL-machine, et voila, we have our desired pup.

In the same way, we would use STR$(I) to manipulate 'I' in ways which would be difficult, if not impossible, to do directly with 'I'. For example :

```
10 I = 123.45: PRINT "You have $";I   (let's forget PRINT USING for now)
```

RUNning this program would produce 'You have $ 123.45', with a space between the '$' sign and the '1', where an imaginary '+' sign resides. Here we have a problem if we wish to produce '$123.45' from '$ 123.45'. Hold on though! Here comes SUPERSTR$ to the rescue! We re-write thus :

```
10 I = 123.45: PRINT "You have $";MID$(STR$(I),2)
```

What we've done here is to convert 'I' to STR$(I) - that is, we've changed 123.45 into a string of ASCII characters corresponding to the individual digits of I - and then used MID$ to print out these characters, commencing with the second, in order to skip the imaginary '+'.

Or, to demonstrate an even trickier example, suppose we wished to reverse the order of the digits, no matter what the length of the number. Here's how we'd go about it :

```
10 J$ = "": INPUT I: I$ = STR$(I)
20 FOR J% = 2 TO LEN(I$)
30 J$ = MID$(I$,J%,1) + J$: NEXT J%: I = VAL(J$)
40 PRINT J$,I: GOTO 10
```

Line 20 starts looping at a count of 2 in order to skip the imaginary '+' sign. Line 30 begins by using J% as an index to scan through I$ one character at a time (from left to right) and to keep sticking the picked-off characters *in front of* what it already has in J$. Finally, it uses our VAL-machine to produce the desired reversed-order number from J$. Line 40 no doubt will give some cause for puzzlement, as it has apparently printed the same number twice. But don't forget that 'J$' is a picture-of-the-number, while 'I' is the actual number itself. Just as your reflection in a mirror (ignoring left-to-right reversal) may look like you, but it's not really you - only an image. Same thing with J$ and I. You can perform arithmetic with I, but not with J$.

VAL comes into its own in XBASIC programs which require numeric responses from the operator. How many times have you accidentally hit a letter (for example, the letter 'O' instead of the digit '0') and had your program bomb with a 'MIXED MODE' error-message? Why not try something like this :

```
10 INPUT "Enter a number ... ",I$: I = VAL(I$)
20 IF I = 0 THEN PRINT "Invalid Entry!": GOTO 10
```

This way your program won't bomb, as an erroneous non-numeric entry for I$ will still be valid, merely producing a VAL of '0', which you trap as an INVALID entry. But what if '0' also happens to be a desired response? We certainly don't want to keep bouncing back to Line 10 in that event! The solution is simply to insert a Line 15 :

```
15 IF I$ = "0" GOTO 30
```

There's only one slight complication here, and that occurs should you enter, say, 12H45 instead of 12345 (so maybe you've had just one teensy-weensy drink too many!). VAL(12H45) would come out as '12', as the conversion process would cease if it comes across a character which is not a digit, a '+', a '-' or a decimal-point. How do we get out of that little hassle? Well, to make sure we've plugged all the loop-holes we would add a new check :

```
25 IF STR$(I) <> I$ THEN PRINT "Invalid Entry!": GOTO 10
```

What we are doing here is to use our instant-camera STR$ to take a picture of 'I' (which could be a genuine '12' or a fake '12' produced from '12H45'). We then compare this picture with our original entry of I$, and if they don't match we know something has gone wrong, so we return for a new entry. Of course, to save a lot of repetition, we would actually re-write Line 20 to read :

```
20 IF I = 0 OR STR$(I) <> I$ THEN PRINT "Invalid Entry!": GOTO 10
```

I'll wager that most readers didn't latch on to the fact that our Line 25 is sufficient unto itself, and that we don't need the check 'IF I = 0' any more! But don't feel too badly about it, as it doesn't exactly stick out like a sore thumb! Study it for a moment and you'll see that our new check actually includes this one in its function.

To Be Continued Next Month

Gespac, Inc

Larry Williams
'68 Micro Magazine
5900 Cassandra Smith Rd
Hixson, TN 37343

Dear Mr. Williams,

I am pleased to announce a successful new addition to our product line for the G-64 bus: A complete single board system based on the 68000/68010 processor, with a large capacity of on-board I/O and memory.

We expect this product to be enormously successful since it combines the most attractive features of our current two best selling boards.

We would be pleased if you could share this important information with the readers of your publication. Enclosed is a press kit we prepared to that end.

Please feel free to call me if you have any questions or need additional information.

Sincerely,

Cosma Pabouctsidis
President

CP/t
Enclosure

In U S A. Contact	In Europe Contact
Cosma Pabouctsidis	Bob White
GESPAC Inc.	GESPAC SA
50 W. Hoover Ave	3 Chemin des Aulx
Mesa, AZ 85202	CH-1228 Geneva
(602) 962-5559	(022) 71 34 00

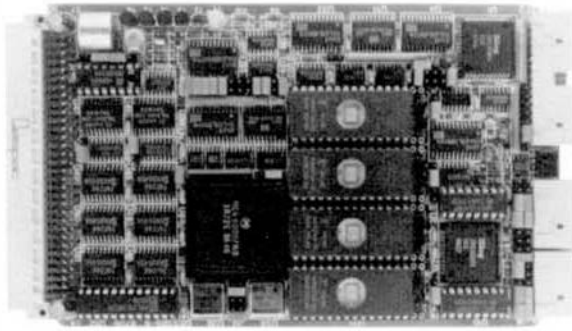
FOR IMMEDIATE RELEASE

GESPAC INTRODUCES HIGH INTEGRATION 68000/68010 SINGLE BOARD SYSTEM FOR THE G-64 BUS

MESA, AZ, APRIL 24, 1987--GESPAC introduces a breakthrough in board density and functionality. The GESSBS-6 is a 68000 based (68010 optional) single board microcomputer system, featuring sufficient EPROM, RAM, and I/O capacity to be used stand alone in a number of imbedded process control and instrumentation applications. The GESSBS-6 is built on a standard single height Eurocard of 100 by 160 millimeters and is fully expandable through the G-64 bus.

The G-64 bus is an easy-to-interface non-multiplexed 16-bit bus aimed at mid range industrial OEM applications. The G-64 bus is an open architecture supported by 40 independent manufacturers, of which 5 are located in North America.

The GESSBS-6 uses a 68000 or 68010 16/32-bit microprocessor running at 8 or 16 MHz. When using the 68010, the board is approximately 40% faster than with the 68000. The board has four 32-pin JEDEC sockets capable of holding 4 EPROMs with capacity up to 512 kilobytes. The sockets can also be configured with two



EPROMs and two CMOS RAM chips for a total capacity of 256 kilobytes of EPROM and 64 kilobytes of RAM.

The GESSBS-6 is equipped with 256 kilobytes of zero-wait-states CMOS RAM. The RAM is automatically powered from the external battery backup in case of power failure. In this configuration, the RAM is made of 8 surface-mounted devices located underneath the EPROM/RAM sockets. The GESSBS-6 can optionally be purchased without the RAM for use in the simplest ROM-based applications. The GESSBS-6 can address up to 8 Megabytes of external memory on the G-64 bus.

The GESSBS-6 is also equipped with two RS-232 compatible serial communication ports. The first port is capable of operating in an asynchronous mode at speeds up to 38,400 baud, and bit or byte synchronous protocols such as IBM Bisync, SDLC and HDLC at speeds of up to 800 kilobits per second. The second port is provided for use in asynchronous mode only at speeds up to 38,400 baud.

The GESSBS-6 is also capable of controlling external digital devices through the use of two double buffered 8-bit parallel I/O ports with four handshake lines. There are three 16-bit timers which can be combined to form a single 32-bit timer. The board also includes a watchdog timer, which, when enabled, will reset the CPU in case of program failure, thus ensuring correct operation under all circumstances.

A Real Time Clock/Calendar is provided on the board to allow the user's program to keep track of the day and time. The clock device is powered by an on-board lithium battery.

The GESSBS-6 is supported with Microware Corp's OS-9 "Unix Like" Real-Time, Multitasking disk operating system. This operating system provides an ideal software development environment for real time industrial systems. OS-9 is modular and can be ROMable in a diskless system.

The GESSBS-6 is available today for $750 with an 8 MHz 68000 CPU without RAM. The 8 MHz version with 256 kilobytes of RAM is available for $995. Prices are for 50 pieces quantity orders.

MOTOROLA INC.

Microprocessor Products Group
6501 William Cannon Drive West
Austin, Texas 78735-8598

EDITORIAL CONTACT:
Mark Vercruysse
512/928-6804

READER CONTACT:
Colleen Collins
512/440-2123

INQUIRY RESPONSE:
R.Q. Green
P.O. Box 52073
Phoenix, AZ 85072

MOTOROLA ANNOUNCES THE MC68606 MULTI-LINK LAPD CONTROLLER

Austin, Texas, May 11, 1987... Motorola Microprocessor Products Group announces the MC68606 Multi-Link LAPD Controller (MLAPD). The MC68606 is a link level protocol processor for high speed data transfer applications in host computers and intelligent end points and for ISDN signalling applications in packet switching equipment. The MLAPD is the only VLSI device to completely implement the CCITT Q.920/Q.921 Link Access Procedure for the entire bandwidth of a primary rate Integrated Services Digital Network (ISDN) interface. LAPD is the proposed protocol for use at the link level (ISO-level 2) for both signalling and data transfer in ISDN configurations. This VLSI device provides a cost effective solution for ISDN link level processing, while encouraging a universal implementation of the LAPD protocol.

The MC68606 is compatible with AT&T specification for ISDN devices and was developed with information supplied by AT&T.

COST EFFECTIVE ISDN SOLUTION

Designed in 1.5 micron HCMOS, the MC68606 features low power consumption, as well as high performance, with serial data rates in excess of 2.048 megabits per second (Mbps). This intelligent communications protocol controller was designed to be used in high speed data transfer applications between host computers and workstations, taking advantage of the bandwidth provided by H11 and H12 channels and the capabilities of ISDN compatible packet switching equipment. The MLAPD is also ideal for use inside the ISDN switch controller as a link level signalling processor. The bandwidth performance of this VLSI device will allow products to realize significant cost savings by moving away from the 64Kbps channelized approach for primary rate bandwidth allocation originally dictated by analog technology in favor of the better utilization offered by allocation of the entire primary rate bandwidth on a demand basis as digital data switching capabilities evolve.

UTILIZES UNIQUE "MODULAR DESIGN APPROACH"

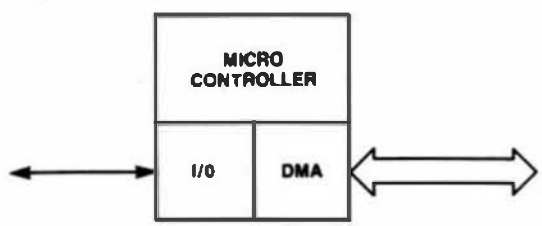
Based upon a proven, modular serial communications processor design, the MLAPD is the third product in a series of serial processor units (SPU) from Motorola (see attached SPU diagram), including the MC68605 X.25 Protocol Controller and the MC68824 Token Bus Controller. As a powerful microcoded engine, the MLAPD provides simultaneous link level control on a context switched, frame by frame basis for up to 8192 logical links using a memory-based architecture. On-chip Direct Memory Access (DMA) bus master capability provides efficient transfer of frames and status information to and from memory. The MLAPD is the first Motorola peripheral to offer the full Motorola or the full Intel system bus interface, selectable at power up. Coupled with support for both 8- and 16-bit data bus configurations and direct addressing of up to 16 megabytes of system memory, the MLAPD is easily tailored to any microprocessor system design.

EXTRA FEATURES INCREASE PERFORMANCE AND FLEXIBILITY

Although the MLAPD implements a shared memory based architecture, the MLAPD also contains a large internal RAM to minimize external memory requirements and to limit system bus utilization. External memory requirements are further reduced by performing a translation from the data link connection identifier contained in the frame address field to a logical link identification number with local significance which is used in the link level processing. When a system supports 16 logical links or less, this translation is performed by on-chip content addressable memory (CAM).

The MLAPD maintains the flexibility of LAPD firmware implementations by supporting programmable protocol parameters and network configurations. The memory-resident, linked receive and transmit structures coupled with the maskable MLAPD interrupts support a wide variety of buffer management schemes. With optional sharing of receive buffer pools among multiple logical links, the user can efficiently allocate memory for active links based upon expected link activity and statistical engineering methods. On the transmit side the user determines the MLAPD servicing of queued information bearing frames, allowing the system to tailor link level handling based upon the link's application.

The MLAPD also provides alternate operational modes to address varying applications and system environments. In its memory-to-memory operational mode, the MLAPD provides LAPD frame processing independent of the system's physical level characteristics to address channelized T1 applications and local area network applications. By optionally inhibiting its HDLC framing, the MLAPD may be easily used with a physical level that supports a parallel interface, such as exists on a backplane in a switching network controller or host co uter. The user may also activate logical links and disable MLAPD application of the LAPD procedures to allow LAPD links to be mixed with non-LAPD links over a primary rate facility. Finally, the MLAPD device may be programmed to operate in promiscuous receive mode where various protocol analyzer features may be invoked, including time stamping of the received link level frames.



SPU BLOCK DIAGRAM

The Serial Processor Units (SPUs) are designed in functional blocks so that major portions of the chip design can be reused in new configurations, allowing Motorola to quickly offer new chip solutions to the various rapidly emerging communication protocols. A microcoded controller, I/O interface, and DMA section are the three major modules of the SPUs. By changing the microcode and modifying portions of the I/O interface, new protocols can be implemented in silicon.

MOTOROLA INC.

Microprocessor Products Group
6501 William Cannon Drive West
Austin, Texas 78735-8598

EDITORIAL CONTACT:
Mark Vercruysse
512/928-6804

READER CONTACT:
Rhonda Alexis Dirvin
512-440-2140

INQUIRY RESPONSE:
R.O. Green
P.O. Box 52073
Phoenix, AZ 85072

MOTOROLA ANNOUNCES THE MC68KTBFA, A SOFTWARE TOOL FOR MANUFACTURING AUTOMATION PROTOCOL (MAP) NETWORKS

Austin, Texas, May 11, 1987... Motorola Microprocessor Products Group announces the MC68KTBFA, a real time software tool that speeds development of token bus networks. The Token Bus Frame Analyzer (TBFA) can keep track of statistics while monitoring network performance, and show specific user chosen frames via an easy to use triggering mechanism. The TBFA can store and display frames and trigger based on any International Standard Organization (ISO) header or any segment of the media access control (MAC) data unit. Connecting a TBFA to a token-passing bus LAN is transparent to the LAN's operation because the TBFA is not part of the token-passing logical ring. The TBFA is easy to use, based on a menu-driven format. All the required information is displayed on the screen at all times, so no paper is required.

The TBFA is a software package that resides on four EPROMs which operate on a Motorola MVME372 MAP interface module board. The MVME372 running the TBFA software is completely stand-alone and requires no backplane bus. The only other equipment required to make the TBFA operational is a modem that matches the LAN, a VT100 terminal or equivalent, and a power supply. All modem connections are through the IEEE 802.4G recommended standard interface.

The cost-effective TBFA, selling for only $2500, will be available June 1, 1987. Discounts are available for multiple copies.

ESP Electronic Specialists, Inc.
171 South Main Street, Natick, Mass. 01760
(617) 655-1532

PRESS RELEASE

EXTENDED RANGE FILTER SUPPRESSOR

Electronic Specialists announces a KLEEN LINE CLASSIC, extended range filter/suppressor for micro computer protection. Featuring 39,000 Surge Amps, 413 Joule protection, the Kleen Line Classic also incorporates a wide band, balanced-pi filter to help eliminate AC Line electrical noise and interference.

Options available include master equipment controlled Delayed Power, Remote Switch to permit desktop activation and Power Fail Interrupt for automatic shut-down in the event of AC power loss.

Model KLC-SFK-36, 10 Amp, 6 socket with 6 foot cord at $71.95. From Stock.

Electronic Specialists, Inc.
171 South Main Street
Natick, Mass. 01760
800-225-4876

Dear Editor:

Here is a check for $49.00 (that is, 2 X $24.50), for two one-year subscriptions to 68 Micro Journal. Please send one year's worth of issues to me at the address above. Please simultaneously send one year's worth of issues to the following person:

Mr. Sig Hartmann
Atari Corporation
1196 Borregas Avenue
Sunnyvale, California 94086

Incidentally, If you or anyone you know would like to undertake a nearly impossible job, you or she or he may wish to write to Mr. Hartmann at the address above to offer your, her, or his services in selling the Atari ST series of computers to VARs or businesses. Atari bears as much "toy" stigma as the CoCo, and it has no "AtariShack" stores to help it reach consumers.

Sincerely,
Bill Lynch

William C. Lee
6778 Bullock Dr.
San Diego, CA. 92114

Dear Mr. Williams:

I just wanted to drop a note to let you know that I've been enjoying the series on Forth. These days, when I pull out the trusty Gimix 6809, it's usually to do a process control test or some highly interactive application, for which Forth is quite well suited. I like seeing others' insights and solutions--there's almost always a clue for everyone.

I've offered a few assembly tidbits in the past, but never taken the time to say thanks for years of timely and useful 68xxx information. I use (from necessity) all the newer machines, but in most ways the Motorola processors outperform them--even the 'ancient' 6809.

I'll always be proud to be one of those diehards Ron Anderson mentioned that remember SWTPC et all as name brands.

Anyway, thanks again, and I look forward to years more.

Sincerely,
William C. Lee

P.S. What kind of 68000 type articles do you look for from the readership?

Editor's Note: Thanks Bill for the thoughts. I guess we all are diehards, I know we are. We still use strictly 68XXX systems for ALL our applications here at CPI. There has not come up one application that we have had to go to a "foreign" computer system. Everything we need doing we can do with our own type systems. And programs like your "SOLVE" do much towards making all that a reality.

As to 68XXX programs - most anything that our readers can learn or benefit from. Thousands of us will appreciate your efforts. Thanks again.

DMW

Re: TIME.CMD - Correction

Dear Mr. Williams:

Thank you for publishing my article on TIME.CMD in your April 1987, issue.

There is a perverse law somewhere that says that as soon as something is published, you find errors in it. I found an error in the code contained in the article. Unless it is corrected, it will cause FLEX to lock up if the time is set after it has been set once before. The problem is caused by the program's failure when the clock chip is being set to check whether the update code has already been linked to FLEX.

As written, WHENEVER the clock chip is set, the program moves the update code out of the utility command space and links it to FLEX. This is what you want it to do if it has not already been linked to FLEX. However, if it has already been linked to FLEX, the second linking causes the loss of the original address of the DWARMS jump in FLEX's disk drivers. This makes it impossible to properly uninstall the date update code. More important, it sets the jump at the end of the update code to point at the update code itself. The result is an endless loop. While this will keep the FLEX date registers current forever, it prevents use of the computer.

In order to fix this problem, the program should be modified as shown below. The added or modified lines are marked with a percent sign (%). The added code checks to see if the update code has been moved out of the utility command space BEFORE it moves the code and links it to FLEX. If the code has already been moved, the program skips the instructions which would move it and link it again.

I am sorry for the inconvenience which this bug caused anyone. These modifications should fix it.

Sincerely,
Ken Drexler
Ken Drexler
365 Drake's View Dr.
Inverness, Calif. 94937

Corrections to Source Code for TIME.CMD

```
* IF RAMLOC ZERO, MOVE MEMEND
 BNE TIME9 SKIP IF ADDR. NOT EQ. 0
 LDD MEMEND GET OLD MEMEND
 SUBD ENDMOV-BEGMOV
 STD MEMEND SAVE RESULT
 TFR D,Y MOVE DESTINATION TO Y

* CHECK IF TIME CODE IS ALREADY MOVED  %
TIME9 LDD ,Y GET CODE AT DESTINATION  %
 CMPD SETSYS ALREADY SET?  %
 BEQ TIME10 YES, SKIP MOVE  %
 PSHS Y SAVE LOCATION OF CODE  %
TIME91 LDA ,X+ MOVE DATA
 STA ,Y+
 CMPX 2,S DONE?
 BNE TIME91
 PULS X,Y GET DESTINATION, CLEAN STACK
 STX DWARM+1 STORE LOC. OF CODE IN DWARM OPERAND
TIME10 LBSR PRDATE PRINT DATE/TIME
EXIT JMP WARMS
```

ACT Applied Computer Technology, Inc.

Mr. Larry Williams
68 Micro Journal
5900 Cassandra Smith Road
Hixson, TN 37343

Dear Larry:

Enclosed is the documentation and new program disk for version 2.2 of our BTree Routines. This change corrects a potential problem I discovered while converting BTree to the C language. To my knowledge, no user has ever found this problem.

The problem only occurs on an OS9 level two system when two or more users are simultaneously accessing the same file. If one user is sequentially stepping through the file with "Next" or "Previous" calls and another user adds or deletes a key in the same block the first user is stepping through, the first user may miss a key or see the same key twice.

As you can see, the problem is not very likely to occur, but I certainly don't want to leave any known problems in the package. Unfortunately, the change is rather extensive and cannot be fixed with just an errata sheet or some similar method. In addition, the documentation has been changed.

Sincerely,
J. Larry Hinsley
J. Larry Hinsley

Barton Laboratories INTELLIGENT TOOLS FOR INTELLIGENT CONTROLS

WEST 14TH STREET AUSTIN, TX

NEW PRODUCT ANNOUNCEMENT

A single-board computer called the LAB 6809 is designed to serve as prototyping tool in the development of applications which use the 6809 microprocessor. This product is unique in providing wirewrapping area for I/O on the same board as the microprocessor, thus allowing maximum performance without buffer delays.

On-board features include jumper selection of 1 or 2 MHz operation with provision for substitution of a user crystal, 8K of RAM/ROM/EPROM, an RS-232C serial port with jumper-selectable baud rates from 75 to 38,400 baud, a 4-bit latch which can extend the address range to 1 megabyte on the SS-50 bus, and provision for up to 4 voltage regulators.

A MRDY cycle stretch circuit with synchronous clocking allows use of slow EPROMS (350 or 450ns) at 2 MHz with minimum delay of 1/2 cycle (250 ns). Synchronous clocking eliminates latch-up problems with the internal 6809 MRDY flip-flop.

The 9.5" by 6.25" board has silkscreened labels on both sides to facilitate wirewrapping and testing. The large wirewrapping area (22 sq. in.) is provided with buffered address, data and control signals, including five independent select lines on 16-byte boundaries, similar to SS-30 slot selection. Signals are arranged to accomodate a second 6850 ACIA or 6840 PTM, and a 6821 PIA. The back side of the board accomodates 78xx and 79xx voltage regulators and heat sinks.

The PICOBUG monitor has 17 commands and 24 I/O subroutines to simplify downloading and debugging. A standard 5.25" diskette provides source and object code for a DOWNLOAD utility for Flex users. (Flex is a trademark of Technical Systems Consultants, Inc.) Comprehensive documentation (92 pages) describes operation and modification options in detail, and includes source code for PICOBUG and DOWNLOAD.

A ribbon cable with DB-25S conoector, a power cable and all heat sinks are included in the purchase price of $395.

Classifieds

As submitted - No Guarantees

DAISY WHEEL PRINTERS
Qume Sprint 9 - $900 Qume Sprint 5 - $800

HARD DISK 10 Megabyte Drive - Seagate Model #412 $275.

3 - Dual 8" drive enclosure with power supply. New in box. $125 each.

5 - Siemens 8" Disk Drive . $100 each.

Tano Outpost II, 56K, 2 5" DSDD Drives, FLEX, MUMPS, $495.

TELETYPE Model 43 PRINTER - with serial (RS232) interface and full ASCII keyboard. $250 ready to run.

SWTPC S/09 with Motorola 128K RAM, 1-MPS2, 1-Paralell Port, MP-09CPU Card - $900 complete.

(615) 842-4600 M-F 9AM to 5PM EST

***

68000 Development System Run Sk Star Dos, Etc.-$100 and up-
914-297-7686

Ωmega . . . The Desktop Mainframe

* 12.5 MHz MC68020 32-bit microprocessor & MC68881 FPCP as standard.
* 1 megabyte of non-volatile, zero wait-state STATIC RAM and 128k/256k bytes ROM.
* 25.5 megabyte (unformatted) Winchester Hard disc as standard.
* 1.0 megabyte (unformatted) DS/DD 80 track floppy disc drive (specify 5¼" or 3½").
* Five RS-232C serial ports (one may be configured as an RS-485 network node).
* One Parallel printer port and one 16-bit bi-directional parallel port.
* Non-volatile clock calendar and power supply supervisor.
* OS-9/68K™ 'Professional' with full MC68881 FPCP support and 'C' compiler.

OMEGA/12.5/OS	OMEGA w/12.5 MHz 020/881 & OS-9/68K™	$4750.00
OMEGA/16.6/OS	OMEGA w/16.67 MHz020/881 & OS-9/68K™	$4950.00
OMEGA/1MB	1 megabyte additional STATIC RAM	$ 895.00
OMEGA/SER9	9 port RS-232C serial expansion board	$ 645.00
OMEGA/GRF	640 x 480 x 4 bits/pixel ACRTC graphics I/F	$ 925.00

PRICES INDICATED INCLUDE SHIPPING AND APPLY TO U.S. CUSTOMERS ONLY

VISA/MASTER CARD ORDERS ACCEPTED

OS-9 is a trademark of Microware Systems Corporation

NON-U.S. CUSTOMERS SHOULD CONTACT THE NEAREST DEALER FOR PRICE & DELIVERY INFORMATION

NORTH & SOUTH AMERICA

LLOYD I/O INCORPORATED
19535 NE GLISAN
P.O. Box 30945
PORTLAND, OR 97230 (USA)
TEL: (503) 666-1097
TLX: 9103805448 LLOYD I O

ALL OTHER ENQUIRIES

WINDRUSH MICRO SYSTEMS
WORSTEAD LABS.
N. WALSHAM, NORFOLK
NR28 9SA, ENGLAND
TEL: (0692) 404086
TLX: 975548 WMICRO-G

CONTINENTAL EUROPE

SNIJDER MICRO SYSTEMS
SCHOOTEINDSEWEG 8a
5756 BD Vlierden
The NETHERLANDS

TEL: (0)4930-11975-13666

THE 6800-6809 BOOKS

..HEAR YE.....HEAR

OS-9™ User Notes

By: Peter Dibble

The publishers of 68' Micro Journal are proud to make available the publication of Peter Dibbles OS9 USER NOTES

Information for the BEGINNER to the PRO, Regular or CoCo OS9

Using OS9
HELP, HINTS, PROBLEMS, REVIEWS, SUGGESTIONS, COMPLAINTS, OS9 STANDARDS, Generating a New Bootstrap, Building a new System Disk, OS9 Users Group, etc.

Program interfacing to OS9
DEVICE DESCRIPTORS, DIRECTORIES, "FORKS", PROTECTION, "SUSPEND STATE", "PIPES", "INPUT/OUTPUT SYSTEM", etc.

Programming Languages
Assembly Language Programs and Interfacing; Basic09, C, Pascal, and Cobol reviews, programs, and uses; etc.

Disks Include
No typing all the Source Listings in. Source Code and, where applicable, assembled or compiled Operating Programs. The Source and the Discussions in the Columns can be used "as is", or as a "Starting Point" for developing your OWN more powerful Programs. Programs sometimes use multiple Languages such as a short Assembly Language Routine for reading a Directory, which is then "piped" to a Basic09 Routine for output formatting, etc.

BOOK $9.95

Typeset -- w/ Source Listings
(3-Hole Punched; 8 x 11)
Deluxe Binder - - - - - - - - - - - $5.50

All Source Listings on Disk

1-8" SS, SD Disk - - - - $14.95
2-5" SS, DD Disks - - - $24.95

FLEX™ USER NOTES

By: Ronald Anderson

The publishers of 68 MICRO JOURNAL are proud to make available the publication of Ron Anderson's **FLEX USER NOTES**, in book form. This popular monthly column has been a regular feature in 68' MICRO JOURNAL SINCE 1979. It has earned the respect of thousands of 68 MICRO JOURNAL readers over the years. In fact, Ron's column has been described as the 'Bible' for 68XX users, by some of the world's leading microprocessor professionals. The most needed and popular 68XX book available. Over the years Ron's column has been one of the most popular in 68 MICRO JOURNAL. And of course 68 MICRO JOURNAL is the most popular 68XX magazine published.

Listed below are a few of the **TEXT** files included in the book and on diskette.

All TEXT files in the book are on the disks.

LOGO.C1	File load program to offset memory — ASM PIC
MEMOVE.C1	Memory move program — ASM PIC
DUMP.C1	Printer dump program — uses LOGO — ASM PIC
SUBTEST.C1	Simulation of 6800 code to 6809, show differences — ASM
TERMEM.C2	Modem input to disk (or other port input to disk) — ASM
M.C2	Output a file to modem (or another port) — ASM
PRINT.C3	Parallel (enhanced) printer driver — ASM
MODEM.C2	TTL output to CRT and modem (or other port) — ASM
SCIPKG.C1	Scientific math routines — PASCAL
U.C4	Mini-monitor, disk resident, many useful functions — ASM
PRINT.C4	Parallel printer driver, without PFLAG — ASM
SET.C5	Set printer modes — ASM
SETBAS1.C5	Set printer modes — A-BASIC

NOTE: .C1,.C2, etc.=Chapter 1, Chapter 2, etc.

**Over 30 TEXT files included is ASM (assembler)-PASCAL-PIC (position independent code) TSC BASIC-C, etc.

Book only: **$7.95** + **$2.50 S/H**

With disk: 5" **$20.90** + **$2.50 S/H**

With disk: 8" **$22.90** + **$2.50 S/H**

Shipping & Handling $3.50 per Book, $2.50 per Disk set
Foreign Orders Add $4.50 Surface Mail
or $7.00 Air Mail
If paying by check - Please allow 4-6 weeks delivery
* All Currency in U.S. Dollars

Continually Updated In 68 Micro Journal Monthly

Computer Publishing Inc.
5900 Cassandra Smith Rd.
Hixson, TN 37343

VISA

(615) 842-4601
Telex 5106006630

"FLEX is a trademark of Technical Systems Consultants
"OS9 is a trademark of Microware and Motorola
"68' Micro Journal is a trademark of Computer Publishing Inc.

OS-9 UniFLEX
MUSTANG-020, 68020, 68881 AND MORE
HANDS-ON EXPERIENCE
NOW SK*DOS available

The DATA-Comp Division of Computer Publishing Corporation announces their new and innovative HANDS-ON 68020 computer familiarization two day event. A chance to TRY BEFORE YOU BUY!

For two full days (Monday through Friday - excluding legal holidays) each participant will be furnished the exclusive use of a 68020 computer (MUSTANG-020). Each system will have available native C compilers, BASIC, assembler and other high level languages. Each system will be equipped with the Motorola MC 68881 math co-processor, where applicable.

Each demonstration room will contain not more than two work stations. Each system will be equipped with floppy disk, 20 megabyte winchester technology hard disk, and 2 megabyte of RAM. RAM is partitioned as 690K bytes of RAM disk and 1.2 megabyte of user RAM space.

Participants are encouraged to bring along any source level projects, for evaluation, in C, BASIC or assembler. Call for availibility of other HHLs.

Although this is not a training seminar, Data-Comp personnel are available for assistance and consultation. This event is scheduled for hands-on evaluations of the 68020 CPU, 68881 math co-processor and MUSTANG-020 system, operating in a functional environment.

Transportation to and from the airport and hotel/motel will be provided. Lunch provided both days. Chattanooga airport is serviced by American, Delta, Republic and other airlines.

COST

One person - $375.00

Two persons - $595.00

* Motel single $22.00, double $26.00
Includes satellite TV - convenient to food and shopping

DATA-COMP

A Division of
Comp ter Publishing, Inc.

5900 Cassandra Smith Road
Hixson, Tn 37343
Telephone 615 842-4600
Telex 510 600-6630

Systems available for both OS-9 and UniFLEX. Reservation should be made 15 days in advance. Attendee should initially indicate OS-9, UniFLEX or both. Special facilities available on request. Please write or call for additional information.

NOTE: Both OS-9 and UniFLEX are Unix type operating systems. Each as been enhanced in some aspect or another. Prospective attendees should have some working knowledge or experience with one of these operating systems, to gain full benefit of the session. However, a newcomer will find that it is a simple matter to be fairly proficient in using these systems in the allocated time. Special system instruction available on request. Call or write.

* Hotel/Motel cost are separate cost, not included in the basic cost shown.

Stop!

Get a 25 MegaByte Hard Disk practically FREE - only 1¢

Be Sure to Consider the SPECIAL MUSTANG 1¢ Sale on page 5

Be Sure to Consider the SPECIAL MUSTANG 1¢ Sale on page 5

When it's over, IT'S OVER!

We don't know how long this very, very low price can be maintained, don't miss it!

Data-Comp Div. - CPI

6809<>68XXX UniFLEX

X-TALK

A C-MODEM/Hardware Hookup

Exclusive for the MUSTANG-020 running UniFLEX, is a new transfer program and cable set from DATA-COMP (CPI). X-TALK consist of 2 disks and a special cable, this hook-up enables a 6809 SWTPC UniFLEX computer to port UniFLEX files directly to a 68XXX UniFLEX system.

This is the only currently available method to transfer files, text or otherwise, from a 6809 UniFLEX system to a 68000 UniFLEX system, that we have seen. A must if you want to recompile or cross assemble your old (and valuable) source files to run on a 68000 UniFLEX system. GIMIX users can directly transfer files between a 6809 GIMIX system and our MUSTANG-020 68020 system, or GIMIX 68020 system. All SWTPC users must use some sort of method other than direct disk transfer. The 6809 SWTPC UniFLEX disk format is not readable by most other 68000 type systems.

The cable is specially prepared with internal connections to match the non-standard SWTPC SO/9 DB25 connectors. A special SWPTC+ cable and software is also available, at the same price. Orders must specify which type SWTPC 6809 UniFLEX system they intend to transfer from or to.

The X-TALK software is furnished on two disks. One 8" disk containing the 6809 software and one 5" disk containing the 68XXX software. These programs are also complete MODEM programs and can be used as such, including X-on X-off, and all the other features you would expect from a full modem program.

X-TALK can be purchased with/without the special cables, however, this SPECIAL price is available only to registered MUSTANG-020 owners.

X-TALK, w/cable	**$ 99.95**
X-TALK only	**69.95**
X-TALK w/source	**$149.95**

DATA-COMP
5900 Cassandra Smith Rd.
Hixson, TN 37343

Telephone 615 842-4601
Telex 510 600-6630

Note: Registered MUSTANG-020 owners must furnish system serial number in order to buy at these special low prices.

68 MICRO JOURNAL
Reader Service Disks

Disk- 1 Filesort, Minicat, Minicopy,Minifms, **Lifetime, **Poetry, **Foodlist, **Diet.
Disk- 2 Diskedit w/ inst.& fixes, Prime, *Prmod, **Snoopy, **Football, **Hexpawn,**Lifetime.
Disk- 3 Cbug09, Sec1, Sec2, Find, Table2, Intext, Disk-exp, *Disksave.
Disk- 4 Mailing Program, *Finddat, *Change, *Testdisk.
Disk- 5 *DISKFIX 1, *DISKFIX 2, **LETTER, **LOVESIGN, **BLACKJAK, **BOWLING.
Disk- 6 **Purchase Order, Index (Disk file indx).
Disk- 7 Linking Loader, Rload, Harkness.
Disk- 8 Crtest, Lanpher (May 82).
Disk- 9 Datecopy, Diskfix9 (Aug 82).
Disk-10 Home Accounting (July 82).
Disk-11 Dissembler (June 84).
Disk-12 Modem68 (May 84).
Disk-13 *Initmf68, Testmf68, *Cleanup, *Dskalign, Help, Date.Txt.
Disk-14 *Init, *Test, *Terminal, *Find, *Diskedit, Init.Lib
Disk-15 Modem9 + Updates (Dec. 84 Gilchrist) to Modem9 (April 84 Commo).
Disk-16 Copy.Txt, Copy.Doc, Cat.Txt, Cat.Doc.
Disk-17 Match Utility, RATBAS, A Basic Preprocessor.
Disk-18 Parse.Mod, Size.Cmd (Sept. 85 Armstrong), CMDCODE, CMD.Txt (Sept. 85 Spray).
Disk-19 Clock, Date, Copy, Cat, PDEL.Asm & Doc., Errors.Sys, Do, Log.Asm & Doc.
Disk-20 UNIX Like Tools (July & Sept. 85 Taylor & Gilchrist). Dragon.C, Grep.C, LS.C, FDUMP.C.
Disk-21 Utilities & Games - Date, Life, Madness, Touch, Goblin, Starshot, & 15 more.
Disk-22 Read CPM & Non-FLEX Disks. Fraser May 1984.
Disk-23 ISAM, Indexed Sequential file Accessing Methods, Condon Nov. 1985. Extensible Table Driven. Language Recognition Utility, Anderson March 1986.
Disk-24 68' Micro Journal Index of Articles & Bit Bucket Items from 1979 - 1985, John Current.
Disk-25 KERMIT for FLEX derived from the UNIX ver. Burg Feb. 1986. (2)-5" Disks or (1)-8" Disk.
Disk-26 Compacta UniBoard review, code & diagram, Burlison March '86.
Disk-27 ROTABIT.TXT, SUMSTEST.TXT, CONDATA.TXT, BADMEN.TXT.
Disk-28 CT-82 Emulator, bit mapped.
Disk-29 **Star Trek
Disk-30 Simple Winchester, Dec.'86 Green.
Disk-31 *** Read/Write MS/PC-DOS (SK*DOS)
Disk-32 Hier-UNIX Type upgrade - 68MJ 2/87

NOTE:
This is a reader service ONLY! No Warranty is offered or implied, they are as received by 68' Micro Journal, and are for reader convenience ONLY (some MAY include fixes or patches). Also 6800 and 6809 programs are mixed, as each is fairly simple (mostly) to convert to the other. Software is available to cross-assemble all.

* Denotes 6800 - ** Denotes BASIC
*** Denotes 68000 - 6809 no indicator.

Specify 8" disk $19.50
5" disk $16.95

Add: S/H - $3.50
Overseas add: $4.50 surface - $7.00 Air Mail, USA Dollars

68 MICRO JOURNAL

PO Box 849
Hixson, TN 37343
615 842-4600 - Telex 510 600-6630

IF YOU NEED IT, WE'VE GOT IT!
(OR WE'LL MAKE IT)

Yes, **PERIPHERAL TECHNOLOGY** still sells the FD-2 for SS-50 Bus Computers. And, if you don't need it, we sell other products from Single Board Computers to Systems which should fit your requirements. Custom Hardware Design is also available. Here's a small sample of what we offer:

FD-2 FLOPPY DISK CONTROLLER

- Controls up to four 5¼" Drives
- Runs in 1 or 2 MHZ Systems
- Can be configured for either 4 or 16
- Addresses per I/O Slot (SS30 or SS30C)
- Uses WD2797 Controller Chip (compatible with 1771/179X Controller Chip)
- Hardware and Software compatible with SWTPC DC-4 controllers
- 6800/6809 Flex Drivers available
- SK*DOS Operating System or OS9/6809 Driver package

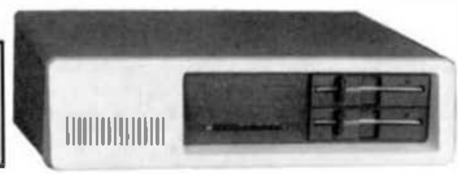
PT69-4 SINGLE BOARD COMPUTER

- 6809E Processor/1 MHZ Clock
- Four RS232 Serial Ports using 6850'S
- Two 8-Bit parallel Ports using 6821 PIA
- Time-Of-Day Clock (MC146818)
- 59K of user RAM
- 2K or 4K of EPROM using 2716 or 2732
- Double Sided/Double Density Floppy Controller
- Can Read/Write Radio Shack OS/9 Diskettes
- Board Size 5.6" X 8.2"

SYSTEMS

- Floppy or Winchester Versions Available
- Systems use any 6809 Single Board Computer or 68008 Board
- Will be configured to meet your Requirements
- OS9 & SK*DOS Operating Systems
- Call or write for system configurations

MONTHLY SPECIAL
FD-2 Floppy Disk Controller: $129.00
Regular Price : $149.95

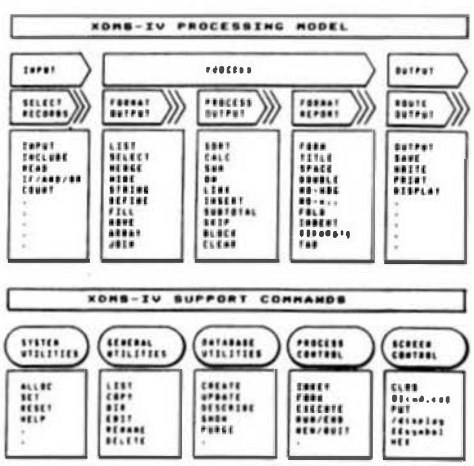


PERIPHERAL TECHNOLOGY
1480 Terrell Mill Road, Suite 870
Marietta, Georgia 30067
(404) 984-0742 Telex # 880584
VISA / MASTERCARD / CHECK / C.O.D.

*OS9 is A Trademark of Microware & Motorola
*SWTPC is A Trademark of Southwest Technical Systems

Catalogues Available Upon Request.

XDMS-IV
Data Management System



Save $100.00 - Limited Time
Regular $350.00 - ~~Now Only~~
$249.95

Technical telephone assistance: Tel 914-941-3552 (Evenings)
FLEX™ Tecnical Systems Consultants, SK*DOS™ STAR-KITS Corp.

FOR 6809 FLEX-SK-DOS(5/8")

Up to 32 groups/fields per record! Up to 12 character filed name! Up to 1024 byte records! User defined screen and print control! Process files! Form files! Conditional execution! Process chaining! Upward/Downward file linking! File joining! Random file virtual paging! Built in utilities! Built in text line editor! Fully session oriented! Enhanced forms! Boldface, Double width, Italics and Underline supported! Written in compact structured assembler! Integrated for FAST execution!

XDMS-IV Data Management System

XDMS-IV is a brand new approach to data management. It not only permits users to describe, enter and retrieve data, but also to process entire files producing customized reports, screen displays and file output. Processing can consist of any of a set of standard high level functions including record and field selection, sorting and aggregation, lookups in other files, special processing of record subsets, custom report formatting, totaling and subtotaling, and presentation of up to three related files as a "database" on user defined output reports.

POWERFUL COMMANDS!

XDMS-IV combines the functionality of many popular DBMS software systems with a new easy to use command set into a single integrated package. We've included many new features and commands including a set of general file utilities. The processing commands are Input-Process-Output (IPO) oriented which allows almost instant implementation of a process design.

SESSION ORIENTED!

XDMS-IV is session oriented. Enter "XDMS" and you are in instant command of all the features. No more waiting for a command to load in from disk! Many commands are immediate, such as CREATE (file definition), UPDATE (file editor), PURGE and DELETE (utilities). Others are process commands which are used to create a user process which is executed with a RUN command. Either may be entered into a "process" file which is executed by an EXECUTE statement. Processes may execute other processes, or themselves, either conditionally or unconditionally. Menus and screen prompts are easily coded, and entire user applications can be run without ever leaving XDMS-IV!

IT'S EASY TO USE!

XDMS-IV keeps data management simple! Rather than design a complex DBMS which hides the true nature of the data, we kept XDMS-IV file oriented. The user view of data relationships is presented in reports and screen output, while the actual data resides in easy to maintain files. This aspect permits customized presentation and reports without complex redefinition of the database files and structure. XDMS-IV may be used for a wide range of applications from simple record management systems (addresses, inventory ...) to integrated database systems (order entry, accounting...). The possibilities are unlimited...

Visa & Master Card Excepted

Telephone: 615-842-4601 or Telex: 510 600-6630
Or Write: S.E. Media, 5900 Cassandra Smith Rd., Hixson, Tenn. 37343